(De-)Constructing TLS 1.3

Markulf Kohlweiss[1], Ueli Maurer[2], Cristina Onete[3], Björn Tackmann[4](✉), and Daniele Venturi[5]

[1] Microsoft Research, Cambridge, UK
markulf@microsoft.com
[2] Department of Computer Science, ETH Zürich, Zurich, Switzerland
maurer@inf.ethz.ch
[3] INSA/IRISA, Rennes, France
cristina.onete@gmail.com
[4] Department of Computer Science and Engineering, UC San Diego, San Diego, USA
btackmann@eng.ucsd.edu
[5] Sapienza University of Rome, Rome, Italy
venturi@di.uniroma1.it

Abstract. SSL/TLS is one of the most widely deployed cryptographic protocols on the Internet. It is used to protect the confidentiality and integrity of transmitted data in various client-server applications. The currently specified version is TLS 1.2, and its security has been analyzed extensively in the cryptographic literature. The IETF working group is actively developing a new version, TLS 1.3, which is designed to address several flaws inherent to previous versions.

In this paper, we analyze the security of a slightly modified version of the current TLS 1.3 draft. (We do not encrypt the server's certificate.) Our security analysis is performed in the constructive cryptography framework. This ensures that the resulting security guarantees are *composable* and can readily be used in subsequent protocol steps, such as password-based user authentication over a TLS-based communication channel in which only the server is authenticated. Most steps of our proof hold in the standard model, with the sole exception that the key derivation function HKDF is used in a way that has a proof only in the random-oracle model. Beyond the technical results on TLS 1.3, this work also exemplifies a novel approach towards proving the security of complex protocols by a modular, step-by-step decomposition, in which smaller sub-steps are proved in isolation and then the security of the protocol follows by the composition theorem.

1 Introduction

SSL/TLS is arguably one of the most widely-used cryptographic protocols securing today's Internet. It was introduced by Netscape [15] in the context of protecting connections between web browsers and web servers, but nowadays the protocol is also used for many other Internet protocols including, e.g., SMTP or IMAP

A. Biryukov and V. Goyal (Eds.): INDOCRYPT 2015, LNCS 9462, pp. 85–102, 2015.
DOI: 10.1007/978-3-319-26617-6_5

(for e-mail transmissions) and LDAP (for accessing directories). Flaws and insecurities in the original design required the protocol to be fixed repeatedly; the current version is TLS 1.2 [12]. A preliminary version of TLS 1.3, which deviates from prior versions considerably, is currently under development [13]. In this paper, we analyze the security of this latest (draft) version of TLS.

1.1 Our Contributions

We prove the security of (a slightly modified version of) the ephemeral Diffie-Hellman handshake of TLS 1.3 with unilateral authentication, that is, where only the server has a certificate. We expect that this mode will be used widely in practice, although recently other modes based on pre-shared keys or Diffie-Hellman with a certified group element have been added to the draft.

More precisely, we prove that TLS 1.3 in ephemeral Diffie-Hellman mode[1] constructs a unilaterally secure channel, that is, a channel where the client has the guarantee that it securely communicates with the intended server while the server has no comparable guarantee. The protocol assumes that an insecure network and a public-key infrastructure (PKI) are available. Our results for TLS 1.3 are in the standard model, with the sole exception that the key derivation function HKDF is used in a way for which security has so-far only been proved in the random-oracle model.[2]

We stress that our result guarantees composability, both in the sense that multiple sessions of the protocol can be used concurrently, and in the sense that the constructed channel can safely be used in applications that assume such a channel. In particular, adding password-based authentication for the client in the unilaterally secure channel immediately yields a mutually secure channel.

Our proof follows a modular approach, in which we decompose the protocol into thinner layers, with easier intermediary proofs. The security guarantee of the entire protocol then follows by composition. In particular:

- Each individual proof consists of a reduction from only a small number of assumptions,[3] and can be updated individually if the corresponding step of the protocol is altered.
- If a better proof is found for one of the smaller sub-steps, re-proving only this sub-step immediately results in an improved security statement of the complete protocol by virtue of the composition theorem.

Modification of the Protocol. While in the original draft [13] the server sends its (PKI) certificate encrypted under preliminarily established keys, we analyze a version of the protocol in which the certificate is sent in clear. Encrypting the certificate complicates the security analysis: on the one hand, the symmetric

[1] Subject to the modification described below.

[2] HKDF is used to extract from a Diffie-Hellman group element without a salt. The only proof of this that we know of relies on random oracles.

[3] The ultimate goal in such a modularization is that the proof of each step consist of only a single reduction, but TLS 1.3 does not allow for this.

keys are authenticated by the certificate (as the latter authenticates the server's key-exchange share); on the other hand, the certificate is protected with the symmetric keys. Our proof can be modified along the lines of a similar analysis of IPsec [16], but at the cost of a more complicated formalization.

Limitations of Our Analysis. Our proof does not cover the notion of (perfect) forward secrecy; the main reason is that no formalization of this property currently exists in the constructive cryptography framework we work in. Note that while our definitions do not model the adaptive corruption of parties, they *do* guarantee that the keys can be used in arbitrary applications, which traditional game-based notions model via so-called key-reveal oracle queries.

Our proof only applies to sessions with a fixed TLS 1.3 version and uses an abstract formulation of a PKI corresponding to the guarantee that (a) a client knows the identity of the server with which it communicates; and (b) only the honest server can get a certificate for this identity [22,24]. This means that some types of attacks are precluded from the model, such as version rollback (by assuming a fixed version) and Triple Handshake [5] (by assuming that the server be honest). This implies, in particular, that our results do not require the collision resistance of the hash function for the security of the key derivation (but only during the authentication); in other words, the additional security achieved by including the session hash into the key derivation is neither defined nor proven. Furthermore, our analysis does not cover session resumption.

Our analysis covers concurrent sessions, at the cost of some complexity in our intermediary proof steps. Indeed, the specific design of TLS makes many of these steps cumbersome by requiring us to model multiple sessions explicitly; this is an effect of TLS breaking natural module boundaries between different parts of the protocol, by explicitly using protocol data from lower levels (i.e., the transmitted messages) in higher-level parts of the protocol (hashed transcripts in the key derivation and the finished messages). Since some of the low-level data used in these computations, such as the server certificate, are correlated across multiple sessions of the same server, we cannot use generic composition to prove them in isolation. In a protocol designed from scratch, one can ensure that the separation of these sessions comes into full effect at a "lower" protocol level, simplifying the proofs for the "higher" levels. Indeed, our difficulties in the analysis encourages constructing protocols that are modular by design and can be analyzed by combining simple modular steps. We stress that even for TLS, we make heavy use of the composition theorem, not only to modularize our analysis, but also to lift the security we obtain for one server and multiple (anonymous) clients to the more standard multiple clients and servers setting, and for composition with arbitrary other protocols.

As most cryptographic work on TLS, we focus on the cryptographic aspects of TLS and many applied concerns are abstracted over. Moreover, as our work is in the constructive cryptography model, with notation yet unfamiliar to our audience, we focused in the body of our submission on the *beauty* (and elegance, within the limits of TLS' design characteristics), rather than the *weight* of our contribution. We invite the interested reader to find the technical details in the full version [17].

1.2 Related Work

On Provable Security. One aspect that is important in modeling and proving security especially of practical protocols is that of *composability*, as cryptographic protocols are rarely designed to be used in isolation. Indeed, a security guarantee in isolation does not necessarily imply security when a proven protocol is part of a larger scheme. While one can generally prove the security of a composite scheme by proving a reduction from breaking any of the component schemes to breaking the composite scheme, security frameworks that allow for *general/universal composition* result in security definitions that relieve one from explicitly proving such a reduction for each composite scheme. Such a reduction immediately follows from the security of the component schemes and the composition theorem.

For instance, suppose that one can prove that a given scheme (e.g. password-based authentication) achieves mutual authentication, assuming that a unilaterally authenticated secure channel already exists. Suppose also that one has several choices of how to construct this unilaterally secure channel, e.g., by RSA or DH-based key-exchange, relying on the existence of a PKI and an insecure network. In this case, the composition theorem implies that one only has to prove that the two candidate schemes construct the unilaterally secure channel; the security of the composition with the password-authentication scheme follows immediately. Frameworks which allow for generic composition are the universal composability framework (UC) due to Canetti [7], the reactive simulatability (RSIM) framework of Pfitzmann and Waidner [23], and the constructive cryptography framework (CC) of Maurer and Renner [20,21], which we use in this work. In particular, one advantage we see in using constructive cryptography is that it describes the way primitives are used within protocols with given resources, and makes explicit the guarantees that they provide in an application context. This provides an indication of how they can be used as part of more complex protocols.

Authenticated Key Exchange. Authenticated key-exchange (AKE) protocols allow two parties to agree on a session key that can be used to secure their communication. The “handshake” of the SSL/TLS protocol can be seen as an AKE. Beyond secure Internet communication, AKE has many other applications, e.g., in card-based identity and payment protocols. The security of AKE protocols was first defined by Bellare and Rogaway [4] as the indistinguishability of real session keys from random keys. However, neither the initial Bellare-Rogaway model, nor its modifications [2,3,8,10] are inherently composable. One special composition of AKE protocols with record-layer-type encryption was shown by Brzuska et al. [6]; however, AKE game-based security is not generally composable. Notions of key exchange in composable frameworks have been defined by Canetti and Krawczyk [9] and by Maurer, Tackmann, and Coretti [22], respectively.

TLS 1.2 vs. 1.3. As the TLS handshake is at present the most prominent AKE protocol, the analysis of its versions up to and including TLS 1.2 has been the subject of numerous papers in the literature. We note, however, that TLS 1.3 has a fundamentally different design from TLS 1.2, which has only been thoroughly analyzed in one publication so far [14]. While elegant and covering all modes in which the TLS 1.3 key derivation is done, this approach follows traditional game-based methods and is neither as modular as ours, nor generally composable. Several parts of the current protocol draft are adapted from work by Krawczyk and Wee [19], this includes the new key derivation scheme that we also describe in Sect. 4 and analyze in [17].

2 Our Approach — Description and Rationale

In constructive cryptography, the (security) guarantees provided to parties in a specific context are formalized in terms of *resources* available to the parties. In our analysis of TLS, resources are typically communication channels or shared secret keys with certain properties. Cryptographic protocols *construct* (desired) resources from assumed resources, and the composition theorem of this framework guarantees that the protocol (using the resources assumed by it) can be used whenever the constructed resource is required (as an assumed resource) in future constructions, i.e., several subsequent constructions can be combined into a single construction.

We model resources as discrete systems that provide one interface to each honest party, along with a specific interface that formalizes the capabilities of a potential attacker. Interfaces are labeled, such as C for a client, S for a server, or E for the attacker. Interfaces can have sub-interfaces (think of them as grouping related capabilities at the same interface for the sake of modularity); we write for instance S/sid for the server sub-interface for session sid. Protocols consist of one protocol engine or *converter* for each honest party. Compared with "traditional" game-based definitions, the *adversary model* corresponds to the capabilities offered via the E-interface at the *assumed resource* and the honest parties' interfaces at the *constructed resource*. For instance, interaction with an insecure network resource corresponds to an active attacker that is in full control of the network (i.e., a chosen-ciphertext attack). The fact that in a constructed channel the messages to be transmitted can be chosen by the distinguisher then corresponds to a chosen-plaintext attack. The *goal* of the game is reflected in the description of the *constructed resource*. The advantage of the *adversary* in game-based definitions corresponds to the advantage of the *distinguisher* in constructive definitions.

Notation. We use a term algebra to describe composite systems, where resources and converters are symbols, and they are composed via specific operations. We read a composed expression starting from the right-hand side resource, extended by systems on the left-hand side. If resource $\mathbf{R}$ has an interface A to which we "connect" a converter α, the resulting system $\alpha^A\mathbf{R}$ is the composition of the two systems, such that the converter connects to the A-interface of the resource $\mathbf{R}$. For resources $\mathbf{R}$ and $\mathbf{S}$, $[\mathbf{R}, \mathbf{S}]$ denotes the parallel composition of $\mathbf{R}$ and $\mathbf{S}$. If we compose a family of resources $(\mathbf{R}_i)_{i \in \{1,\ldots,n\}}$ in parallel, we also write this as a product, e.g. $\bigotimes_{i=1}^{n} \mathbf{R}_i$. We introduce special notation for families of interfaces $\mathcal{L}$ and converters $\alpha_{\mathcal{L}} = (\alpha_\ell)_{\ell \in \mathcal{L}}$. To attach each π_ℓ to interface ℓ of a resource $\mathbf{R}$, we write $(\alpha_{\mathcal{L}})^{\mathcal{L}}\mathbf{R}$.

Constructions. The construction notion is defined based on the distinguishing advantage between two resources $\mathbf{U}$ and $\mathbf{V}$, which can be seen as a distance measure on the set of resources.[4] A distinguisher is a discrete system that connects to all the interfaces of a resource and outputs a single bit. The *distinguishing advantage of a distinguisher* $\mathbf{D}$ *on two systems* $\mathbf{U}$ *and* $\mathbf{V}$ is defined as

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

The two main conditions defining a *construction* are: (1) *availability* (often called correctness), stipulating that the protocol using the assumed resource behaves as the constructed resource if no attacker is present at the E-interface; and (2) *security*, requiring that there exists a simulator, which, if connected at the E-interface of the constructed resource, achieves that the constructed resource with the simulator behaves like the protocol with the assumed resource (w.r.t. the distinguisher). For the availability condition, the "special converter" $\perp$ signals the attacker's absence; this is taken into account explicitly in the description of the resources. Formally a construction is defined as follows:

Definition 1. *Let* ε_1 *and* ε_2 *be two functions mapping each distinguisher* $\mathbf{D}$ *to a real number in* $[0,1]$. *Let* $\mathcal{L}$ *be the interfaces of protocol participants. A protocol* $\pi_{\mathcal{L}} = (\pi_\ell)_{\ell \in \mathcal{L}}$ constructs resource $\mathbf{S}$ from resource $\mathbf{R}$ with distance $(\varepsilon_1, \varepsilon_2)$ and with respect to the simulator σ, *denoted*

$$\mathbf{R} \quad \overset{\pi_{\mathcal{L}}, \sigma, (\varepsilon_1, \varepsilon_2)}{\Longmapsto} \quad \mathbf{S},$$

if, for all distinguishers $\mathbf{D}$,

$$\begin{cases} \Delta^{\mathbf{D}}\left((\pi_{\mathcal{L}})^{\mathcal{L}} \perp^E \mathbf{R}, \perp^E \mathbf{S}\right) \leq \varepsilon_1(\mathbf{D}) & \quad (\textit{availability}), \\ \Delta^{\mathbf{D}}\left((\pi_{\mathcal{L}})^{\mathcal{L}} \mathbf{R}, \sigma^E \mathbf{S}\right) \leq \varepsilon_2(\mathbf{D}) & \quad (\textit{security}). \end{cases}$$

Games. Several of our construction steps are proved by reductions to the security of underlying primitives, which are defined via game-based notions. A game can be seen as a system that, when connected to an adversary, determines a single bit W (denoting whether the game is won or lost). The success probability of an adversary $\mathbf{A}$ with respect to a game $\mathbf{G}$ is

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

[4] The distinguishing advantage is in fact a pseudo-metric on the set of resources, that is, it is symmetric, the triangle inequality holds, and $d(x, x) = 0$ for all x. However, it may be that $d(x, y) = 0$ for $x \neq y$.

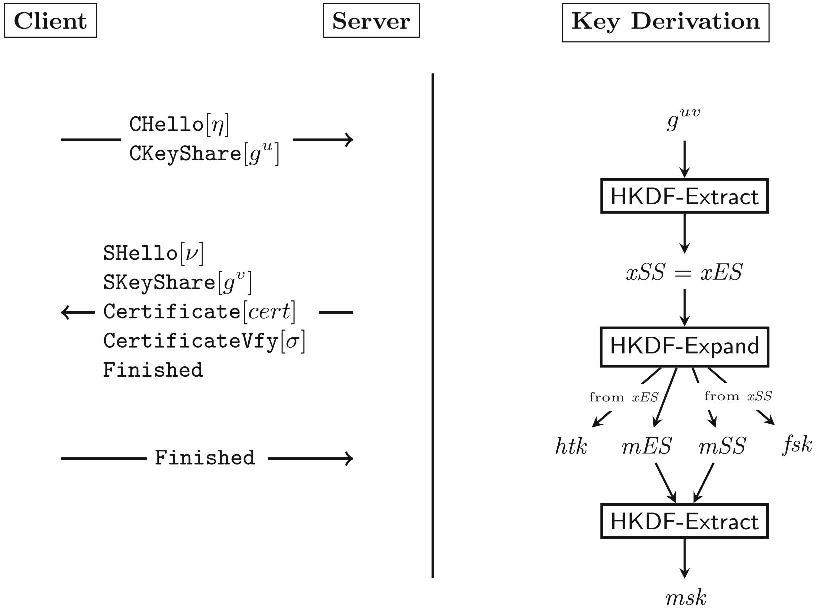

Fig. 1. The TLS 1.3 handshake and the key derivation in the case that the ephemeral and the static handshake secret coincide.

For games that are defined as distinguishing problems (such as IND-CPA security for encryption schemes), we use the notation from Eq. (1), that is, if the game is described by the pair $(\mathbf{G}_0, \mathbf{G}_1)$, then we are interested in the advantage $\Delta^{\mathbf{A}}(\mathbf{G}_0, \mathbf{G}_1)$. Both $\Gamma^{(\cdot)}(\mathbf{G})$ and $\Delta^{(\cdot)}(\mathbf{G}_0, \mathbf{G}_1)$ define adversarial advantage functions $\epsilon(\cdot)$, such as $\varepsilon_{\mathsf{cr}}(\mathbf{A}) = \Gamma^{\mathbf{A}}(\mathbf{G}^{\mathsf{cr}})$ for the collision resistance of a hash function, $\varepsilon_{\mathsf{uf\text{-}cma}}(\mathbf{A}) = \Gamma^{\mathbf{A}}(\mathbf{G}^{\mathsf{uf\text{-}cma}})$ for the unforgeability of a signature, or $\varepsilon_{\mathsf{ddh}}(\mathbf{A}) = \Delta^{\mathbf{A}}\left((g^A, g^B, g^{AB}), (g^A, g^B, g^C)\right)$ for the intractability of the DDH assumption.

3 TLS 1.3 and Unilaterally Secure Channels

The general structure of TLS 1.3 in (EC)DHE mode is depicted in Fig. 1 on the left. The client hello message includes a 32-byte nonce η; the client key share fixes an (elliptic curve) group $\mathbb{G}$ of order $q = |\mathbb{G}|$ (with some generator g), and an element g^u for some $u \leftarrow\!\!\$ \{1, \ldots, q\}$ in that group. The server verifies that the proposed group is in the list of acceptable groups; if so, it chooses a 32-bit nonce ν (the server hello message), and sends this, together with its key share g^v for $v \leftarrow\!\!\$ \{1, \ldots, q\}$, its certificate (in the initial draft, encrypted with the handshake keys, but in our case, without the encryption), and a certificate verify message, namely a signed session hash, also encrypted in the original draft. As a final message, the server sends an encryption (with its handshake transfer key *htk*) of the finished message.

The finished message is computed by evaluating a PRF keyed with the finished secret *fsk* on the session hash. If the signature and finished message verify, the client finished message is computed analogously and sent to the server, completing the handshake.

The current version of key derivation in TLS 1.3[5] uses HKDF[6] as a replacement of the TLS PRF construction that was the backbone of previous versions. This new key derivation, depicted in Fig. 1 on the right, follows a more stringent cryptographic design and adapts easily to various TLS handshake modes, such as the an as-yet underspecified zero round-trip time (0-RTT) mode, in which the client uses a previously-saved configuration to connect to a pre-known server.

While we leave a more technical, detailed description of the key-derivation steps to Sect. 4, note that we focus in this paper on one particular case of the key derivation in which the client and server calculate only one Diffie-Hellman value, obtained from the client and the server ephemeral key shares. The key derivation in the TLS draft is also prepared for cases in which the two parties compute *two* Diffie-Hellman values, one from the client share and the static server share, and another from the same client share and the ephemeral server share. In the case we consider, those two values are defined to be identical.

Unilaterally Secure Transmissions. The goal of TLS with server-only authentication is modeled by the following *unilateral* channel resource $\twoheadleftarrow\ \twoheadrightarrow\!\bullet_n$. This resource is explicitly parametrized by the bound n on the number of sessions in which an attacker uses a specific *client* nonce (this parameter appears in the security bound). Parties input messages of length (at most) equal to TLS's maximum fragment size. We denote the set of all plaintexts as $\mathcal{PT}$.

$\twoheadleftarrow\ \twoheadrightarrow\!\bullet_n$

No attacker present: Behave as a (multi-message) channel for messages in $\mathcal{PT}$ between interfaces C and $S/1$.

Attacker present:

- Upon the *first* input (`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 $m \in \mathcal{PT}$ at the C-interface, output $|m|$ at interface E.
  - On input (`deliver`, `client`) at the E-interface, deliver the next message at S/e.
  - On input a message $m \in \mathcal{PT}$ at the S/e-interface, output $|m|$ at interface E.
  - On input (`deliver`, `server`) at the E-interface, deliver the next message at C.
- After input (`conquer`, e) with $e \in [n]$ at the E-interface (if e was not used before), forward messages in $\mathcal{PT}$ between the S/e- and E/e-interfaces in both directions.

[5] https://tools.ietf.org/id/draft-ietf-tls-tls13-07.txt.

[6] http://www.ietf.org/rfc/rfc5869.txt.

Intuitively, if no attacker is present, then the resource behaves like a direct channel between a client C and a server's $S/1$ sub-interface. If the attacker *is* present, then we have either a secure channel between the client and the server (first input $(\texttt{allow}, e)$) or, if the attacker was the one performing the handshake (input $(\texttt{conquer}, e)$), a channel between the attacker and the server.

The Assumed Resources. The resources we assume for the TLS protocol are: First, an insecure network NET (obtained by using the TCP/IP protocol over the Internet), where the attacker can also learn the contents of messages transmitted over the network, stop them, or inject arbitrary messages of his choice into the network. Second, a public-key infrastructure (PKI) resource, which we view as specific to a single server (whose identity we assume the client knows). This PKI resource allows the server to send one message (its signature verification key) authentically to all clients, thus capturing the guarantee that only the honest server can register a public key *relative to its own identity*, and the clients verify that the certificate is issued with respect to the expected identity. For simplicity, we consider a model where the PKI is local to the security statement; aspects of modeling a global PKI in composable security frameworks are discussed by Canetti el al. [11].

The Security Achieved by TLS 1.3. We show that TLS 1.3 constructs $\twoheadleftarrow \twoheadrightarrow\!\bullet_n$ from PKI and NET by sequential decomposition of the protocol in the main steps (right to left) shown in Fig. 2. At each step, the resources constructed in previous steps are used as assumed resources in order to construct a "new" resource, until we construct the unidirectional channel $\twoheadleftarrow \twoheadrightarrow\!\bullet_n$. We describe these steps in the rest of this paper.

Our reductions use the pseudorandomness of HMAC, as used internally by HKDF, the pseudorandomness of HKDF itself when seeded with seed 0, the unforgeability of signatures, the collision resistance for the hash function, the intractability of the DDH assumption, and the security of authenticated encryption. We write $\varepsilon_{\mathsf{hmac}}, \varepsilon_{\mathsf{kdf}}, \varepsilon_{\mathsf{uf\text{-}cma}}, \varepsilon_{\mathsf{cr}}, \varepsilon_{\mathsf{ddh}}, \varepsilon_{\mathsf{aead}}$ for their advantage functions.

Theorem 2. *Let $\mathcal{C}$ be a set of clients. The TLS 1.3 protocol constructs, for each client $C \in \mathcal{C}$, one unilaterally secure channel $\twoheadleftarrow \twoheadrightarrow\!\bullet_n$ from NET and PKI. Concretely, for the simulator σ and the adversaries $\mathbf{A}_1, \ldots, \mathbf{A}_{11}$ obtained from $\mathbf{D}$ by explicit reductions derived from those in the modular proof steps,*

$$[\mathsf{NET}, \mathsf{PKI}] \quad \xmapsto{(\mathsf{tls13c},\mathsf{tls13s}),\sigma,(\varepsilon_1,\varepsilon_2)} \quad \bigotimes_{(I,J)\in\mathcal{P}} [\![\twoheadleftarrow \twoheadrightarrow\!\bullet_n]\!]^{(I,J)},$$

with:

$$\varepsilon_1(\mathbf{D}) \coloneqq \binom{|\mathcal{C}|}{2} \cdot 2^{-256} + |\mathcal{C}| \left(\varepsilon_{\mathsf{ddh}}(\mathbf{A}_1) + 2\varepsilon_{\mathsf{prf}}(\mathbf{A}_2) + 2\varepsilon_{\mathsf{kdf}}(\mathbf{A}_3) + \varepsilon_{\mathsf{hmac}}(\mathbf{A}_4)\right)$$

and

$$\varepsilon_2(\mathbf{D}) \coloneqq \left(\binom{n}{2} + \binom{|\mathcal{C}|}{2}\right) \cdot 2^{-256} + \varepsilon_{\mathsf{uf\text{-}cma}}(\mathbf{A}_5) + \varepsilon_{\mathsf{cr}}(\mathbf{A}_6) + n|\mathcal{C}| \cdot \varepsilon_{\mathsf{ddh}}(\mathbf{A}_7)$$
$$+ n|\mathcal{C}| \; \left(2\varepsilon_{\mathsf{prf}}(\mathbf{A}_8) + 2\varepsilon_{\mathsf{kdf}}(\mathbf{A}_9) + \varepsilon_{\mathsf{hmac}}(\mathbf{A}_{10})\right) + 2|\mathcal{C}| \cdot \varepsilon_{\mathsf{aead}}(\mathbf{A}_{11}).$$

This statement holds for all distinguishers $\mathbf{D}$*, some injection* $\rho : \mathcal{C} \to \mathcal{N}$*, and* $\mathcal{P} \coloneqq \{(C, S/\rho(C)) : C \in \mathcal{C}\} \cup \{(E/\eta, S/\eta) : \eta \in \mathcal{N} \setminus \rho(\mathcal{C})\}$*.*

In the theorem, we construct the parallel composition $\bigotimes_{(I,J)\in\mathcal{P}} [\![\twoheadleftarrow\ \twoheadrightarrow\!\bullet_n]\!]^{(I,J)}$ with interfaces (I, J) taken from the set $\mathcal{P}$. This models that the server can identify clients only by some value used in the handshake — we chose the random nonce $\rho(C) \in \mathcal{N}$ — and that the attacker can also interact with the server using "new" nonces, picked by none of the clients.

As a corollary and following a result by Tackmann [24], we model the use of password-based authentication to construct a bilaterally secure channel. We assume a password distribution with maximum guessing probability ϵ as an additional resource $\mathbf{Q}$. Then the constructive corollary we postulate and prove in Sect. 5 is:

Lemma 3. *Sending and checking a password constructs from* $\twoheadleftarrow\ \twoheadrightarrow\!\bullet_n$ *the channel* $\bullet\!\twoheadleftarrow\ \twoheadrightarrow\!\bullet$*, for a distribution* $\mathbf{Q}$ *of passwords as described above. More formally, there is a simulator* σ *such that,*

$$[\twoheadleftarrow\ \twoheadrightarrow\!\bullet_n, \mathbf{Q}] \qquad \overset{\mathsf{pwd},\sigma,(0,\epsilon)}{\Longmapsto} \qquad \bullet\!\twoheadleftarrow\ \twoheadrightarrow\!\bullet.$$

4 De-Constructing TLS 1.3

This section acts as a stage-by-stage proof for Theorem 2. Our strategy is to prove that *individual parts* of the TLS protocol *construct* intermediate resources, which can be used as assumed resources for the next modular construction step. At the end, we use the composition theorem to show that the *entire* TLS 1.3 protocol constructs the $\twoheadleftarrow\ \twoheadrightarrow\!\bullet_n$ channel shown in the previous section.

The structure of our proof follows Fig. 2, read from right to left. We begin by constructing a unique name resource, by choosing a random client nonce uniformly at random from the set of 32-byte strings. The unique name resource is then used to name client *sessions* on the insecure network NET; thus, from a constructive point of view, the nonce exchange at the beginning of the TLS protocol constructs from the resources NET and NAME the network-with-sessions resource SNET.

The subsequent two steps construct the handshake key resource DHKEY from the assumed PKI resource and the newly-constructed SNET resource. We proceed as follows: we first use these two resources to construct an authenticated network-transmission resource $\succ\!-\ -\!\bullet$ (the corresponding TLS step is signing the server's first message; its ephemeral share). From this $\succ\!-\ -\!\bullet$ resource, we construct the handshake key resource DHKEY by simply exchanging the client and server shares to calculate a Diffie-Hellman secret.

The next step is then to use the key derivation described in Fig. 1 to extract an (almost) uniformly random bit-string key from the Diffie-Hellman secret, and expand this to obtain all application keys required by the subsequent protocol steps.

The final step of the protocol is the actual payload protection, which begins by exchanging the finished messages computed using derived keys, and subsequently protecting plaintext messages using authenticated encryption.

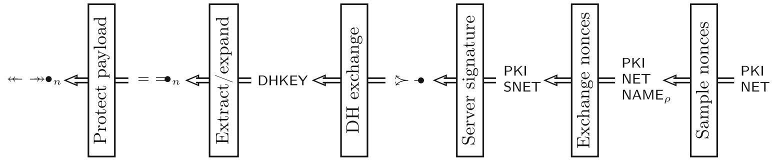

Fig. 2. The decomposition of TLS 1.3.

Session Naming. We formalize unique client naming by means of a resource NAME_ρ, parametrized by an injection ρ from the set $\mathcal{C}$ of honest clients to the set $\mathcal{N}$ of nonces; this resource returns to each client a unique nonce. NAME_ρ can be constructed from scratch: As a nonce contains 256 bits of randomness for TLS 1.3, choosing a nonce at random yields a unique nonce per client up to a loss of $\binom{|\mathcal{C}|}{2}2^{-256}$, where $|\mathcal{C}|$ is the total number of honest clients.

Naming Network Sessions. The client nonce η helps the server associate a session with some client C. Honest clients use distinct nonces, obtained from the NAME_ρ resource; however, an attacker can start many sessions with the same nonce (possibly generated by an honest client). Thus, we index sessions by pairs $sid = (\eta, e) \in \mathcal{N} \times \mathbb{N}$, where e differentiates sessions with the same η. The server's nonce ν for that session is chosen at random and sent to the client; this protocol constructs, from

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

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

No attacker present: For each $C \in \mathcal{C}$ choose a nonce $\nu_{sid} \leftarrow\$\ \mathcal{N}$ with $sid = (\rho(C), 1)$, output $(\rho(C), \nu_{sid})$ at interface C and ν_{sid} at interface S/sid, and provide bidirectional channels between those two interfaces.

Attacker present: Initialize $e_\eta = 0$ for all $\eta \in \mathcal{N}$. For each $C \in \mathcal{C}$, output the nonce η at the interface C and forward all messages input at interface C to the interface E/C. Additionally:

- Upon input $(\mathtt{ack}, \eta)$ at interface E, if $e_\eta \leq n$, then for $sid = (\eta, e_\eta)$ choose a nonce ν_{sid} uniformly at random from $\mathcal{N} \setminus \{\nu_{(\eta,1)}, \ldots, \nu_{(\eta,e_\eta-1)}\}$, output ν_{sid} at interface E/sid and S/sid, and increase e_η. Then, forward all communication between the interfaces S/sid and E/sid.
- Upon input $(\mathtt{deliver}, C, \nu)$ with $C \in \mathcal{C}$, output the server's nonce ν at the interface C. Then, forward all messages between interfaces E/C and C.

Fig. 3. The network resource that additionally outputs nonces.

the resources NAME_ρ and the network resource NET, the resource SNET (the full details and description of the client and server converters, denoted hec and hes_n are left to the full version).

The resource SNET, described in Fig. 3, has interfaces labeled $C \in \mathcal{C}$ for the clients, a server interface S with one sub-interface for each pair (η, e), where $\eta \in \mathcal{N}$ is a nonce, not necessarily from an honest client, and $e \in [n]$ is a counter indicating how many sessions are initiated with nonce η, and an attacker's interface called E. To simplify further construction steps, we rule out collisions for *server* nonces in the SNET resource below, in sessions associated with the same nonce (i.e., $sid = (\eta, e)$ and $sid' = (\eta, e')$). Since the server nonce has the same structure as the client nonce, the security loss is analogous.

The following statement holds:

Lemma 4. *Let $\mathcal{C} \subseteq \mathcal{A}$ and let $\rho : \mathcal{C} \to \mathcal{N}$ be an injective mapping. The protocol $(\mathsf{hec}, \mathsf{hes}_n)$ constructs the resource $\mathsf{SNET}_{\rho,n}$ from the resources NET and NAME_ρ. In more detail, for the simulator σ in the proof:*

$$[\mathsf{NET}, \mathsf{NAME}_\rho] \quad \overset{(\mathsf{hec},\mathsf{hes}_n),\sigma,(0,\varepsilon)}{\Longmapsto} \quad \mathsf{SNET}_{\rho,n},$$

with $\varepsilon(\mathbf{D}) \coloneqq \binom{n}{2} \cdot 2^{-256}$ for all distinguishers $\mathbf{D}$.

The Shared Key Resource. The next step is to construct the Diffie-Hellman key DHKEY; we decompose this step into two smaller steps, briefly described below (we refer to [17] for full details). We represent the DHKEY resource as a particular parametrization of the generic shared key resource $\mathsf{KEY}_{\rho,AUX,n,\mathcal{K}}$ detailed in Fig. 4, with a key space $\mathcal{K}$ that is the Diffie-Hellman group $\mathbb{G}$.

Our first step is to construct from the PKI and SNET resources an authenticated network resource $\succ\!\!-\!\!\bullet_{\rho,\mathfrak{F},SIG,n,h}$ using the certificate and the signature in the TLS certificate verify message. This resource allows the server to transmit one message in each session authentically; this is achieved by signing the message together with a hash of the handshake messages in order to bind it to the session. The reduction relies on the unforgeability of the signature scheme and the collision resistance in the handshake hash.

From $\succ\!\!-\!\!\bullet_{\rho,\mathfrak{F},SIG,n,h}$, we then construct, under the DDH assumption in $\mathbb{G}$, the resource DHKEY. Intuitively, the converters here are simply exchanging the Diffie-Hellman elements and perform the corresponding computation, where the transmission of the server's message relies on the authentication guarantees of the assumed resource. In particular, the signature computed and forwarded in the authentication step allows a client to abort an execution if the signature verification on the handshake hash fails. This is reflected in the second bullet point of the resource $\mathsf{KEY}_{\rho,AUX,n,\mathcal{K}}$.

The composition theorem allows us to combine the two intermediary steps in the following lemma, where we denote by hsc and hss the compositions of the two converters (protocol steps) outlined above:

Lemma 5. *The protocol $(\mathsf{hsc}, \mathsf{hss})$ constructs from the assumed resources PKI and $\mathsf{SNET}_{\rho,n}$ the resource DHKEY, given that: the signature scheme used in certification*

Shared Key Resource $\mathsf{KEY}_{\rho,AUX,n,\mathcal{K}}$

No attacker present: For each $C \in \mathcal{C}$, choose a key $\kappa_C \twoheadleftarrow\!\$\ \mathcal{K}$, a nonce $\nu_{sid} \twoheadleftarrow\!\$\ \mathcal{N}$ (with $sid = (\rho(C), 1)$), and auxiliary information $aux_C \twoheadleftarrow\!\$\ AUX$, and output $(\kappa_C, \rho(C), \nu_{sid}, aux_C)$ at interface C and $(\kappa_C, \nu_{sid}, aux_C)$ at the interface S/sid. Then, provide bidirectional channels between those two interfaces.

Attacker present: Initialize $e_\eta = 1$ for all $\eta \in \rho(\mathcal{C})$.

- Upon input $(\texttt{ack}, \eta)$ at the E-interface, if $\eta \in \mathcal{N}$ and $e_\eta \leq n$, then choose ν_{sid} uniformly at random from $\mathcal{N} \setminus \{\nu_{(\eta,1)}, \ldots, \nu_{(\eta,e_\eta-1)}\}$ for $sid = (\eta, e_\eta)$, output ν_{sid} at the E-interface and increase e_η.
- Upon input $(\texttt{allow}, C, aux, \nu)$ at the E-interface with $aux \in \{0,1\}^*$, and $\nu \in \mathcal{N}$:
  1. If $\nu = \nu_{sid}$ and there was a previous $(\texttt{server-allow}, \eta, e, aux)$ resulting in setting κ_{sid} for some $sid = (\eta, e)$ with $\eta = \rho(C)$ and $e < e_\eta$, then set $\kappa_C = \kappa_{sid}$; else, abort without generating keys.
  2. Output at the C-interface $(\kappa_C, \rho(C), \nu, aux)$.

  Afterward, if at the C-interface a message m is input, output m at the E-interface. Also, allow at the E-interface to inject messages m', to be output at the C-interface.
- Upon input $(\texttt{server-allow}, \eta, e, aux)$ at the E-interface with $e < e_\eta$, with $sid = (\eta, e)$:
  1. If, for $C = \rho^{-1}(\eta)$, $(\texttt{allow}, C, *, \nu_{sid})$ was input at the E-interface before (i.e., with the current server's nonce), then set $\kappa_{sid} = \kappa_C$; else draw κ_{sid} uniformly at random from $\mathcal{K}$.
  2. Output $(\kappa_{sid}, \nu_{sid}, aux)$ at interface S/sid.
- Upon input $(\texttt{inject}, \eta, e, aux, \kappa)$ at the E-interface, if $e < e_\eta$, then for $sid = (\eta, e)$ output (κ, ν_{sid}, aux) at interface S/sid.

After either a $(\texttt{server-allow}, \eta, e, *)$ or a $(\texttt{inject}, \eta, e, *, *)$ message, if at interface S/sid (with $sid = (\eta, e)$) a message m is input, output m at interface E. Also, allow at interface E to inject messages m' to be output at interface S/sid.

Fig. 4. The shared key resource.

is unforgeable, the hash function is collision resistant, and the DDH assumption holds. More formally, for the simulator σ and the reductions $\mathbf{C}_1, \ldots \mathbf{C}_4$ described in the proof,

$$[\mathsf{SNET}_{\rho,n}, \mathsf{PKI}_{\mathfrak{F}}] \quad \overset{(\mathsf{hsc},\mathsf{hss}),\sigma,(\varepsilon_1,\varepsilon_2)}{\Longmapsto} \quad \mathsf{DHKEY}_{\rho,AUX,n},$$

such that for all distinguishers $\mathbf{D}$: $\varepsilon_1(\mathbf{D}) \coloneqq |\mathcal{C}| \cdot \varepsilon_{\mathsf{ddh}}(\mathbf{DC}_1)$, and $\varepsilon_2(\mathbf{D}) \coloneqq \varepsilon_{\mathsf{uf\text{-}cma}}(\mathbf{DC}_2) + \varepsilon_{\mathsf{CR}}(\mathbf{DC}_3) + n \cdot |\mathcal{C}| \cdot \varepsilon_{\mathsf{ddh}}(\mathbf{DC}_4)$.

Expanding the Key. The next step is to extract from the Diffie-Hellman secret and then expand the keys (following the scheme shown in Fig. 1). Finally, the finished messages used for key confirmation are computed. Interestingly, the only effect of the finished messages in our case is that the client and server detect mismatching keys before the first application data is accepted by the protocol. This does not exclude, however, that these messages serve a more crucial role in certain handshake modes or for proving specific security properties we do not consider in this paper.

The key derivation in the newest draft of TLS 1.3 differs considerably from that of TLS 1.2. From the Diffie-Hellman secret, several sets of session keys are derived for use in symmetric primitives: the application traffic keys *atk* for the protection of the payload data, handshake traffic keys *htk* used to protect some data packets in the handshake, the finished secret *fsk* used for the finished messages, and early-data keys used in the 0-RTT mode (the latter do not appear in our analysis). All computations are based on HKDF [18].

The key derivation can be described in several steps corresponding in our analysis to separate, simple construction steps that are composed via the composition theorem:

1. First, two keys xES and xSS are computed by calling $\mathsf{HKDF}.\mathit{extract}(0, \mathit{pmk})$, that is, evaluating the HKDF extraction with seed 0 on the Diffie-Hellman key *pmk* computed in the key exchange. This step assumes the security of HKDF as a computational extractor (therefore relying on a statement proven in the random-oracle model).
2. Using the expansion of HKDF, several keys are computed:
   (a) The finished secret $\mathit{fsk} \leftarrow \mathsf{HKDF}.\mathit{expand}(xSS, \text{“finished”}, h)$ for the confirmation messages, where h is the hash of the handshake messages,
   (b) the “static” master secret value $mSS \leftarrow \mathsf{HKDF}.\mathit{expand}(xSS, \text{“static”}, h)$,
   (c) the “ephemeral” value $mES \leftarrow \mathsf{HKDF}.\mathit{expand}(xES, \text{“ephemeral”}, h)$,
   (d) the handshake traffic keys $\mathit{htk} \leftarrow \mathsf{HKDF}.\mathit{expand}(xES, \text{“handshake”}, h)$.
   This step assumes the security of the HKDF expansion as a pseudo-random function.
3. Then, compute the master secret key $\mathit{msk} \leftarrow \mathsf{HKDF}.\mathit{extract}(mSS, mES)$ by using HKDF to extract from mES using the seed mSS. This step relies only on the fact that the HKDF extraction is a pseudo-random function, as mSS is a good key — in fact a weak PRF is sufficient as mES is (pseudo) random.
4. Expand the application traffic keys *atk* by an HKDF expansion as follows: $\mathit{atk} \leftarrow \mathsf{HKDF}.\mathit{expand}(\mathit{msk}, \text{“application”}, h)$. This step again relies on the HKDF expansion being a PRF.

In order to treat the expanded keys as separate resources for each client, we also incorporate the generation of the finished messages into the construction of those keys. Those messages are computed by evaluating HMAC with the key *fsk* on the session hash h and static labels. This requires that HMAC is a PRF. Since the expansion is the final step that explicitly relies on values that are consistent across several sessions (such as the server's certificate), the constructed expanded-key resource $=\!=\!\!\bullet_n$ can be described in a way that is single-client, as opposed to the more complicated $\mathsf{KEY}_{\rho,AUX,n,\mathcal{K}}$ resource. The resource $=\!=\!\!\bullet_n$ allows a single client and server session to compute the same keys and finished messages if the attacker did not establish that server session himself. Otherwise, the server and attacker share keys, as depicted in the description of $=\!=\!\!\bullet_n$ [17]. We describe the resource we want to obtain at key expansion by: $\bigotimes_{C\in\mathcal{C}}\llbracket = =\!\!\bullet_n\rrbracket^{(C,S/\rho(C))}$, i.e. a parallel composition of such channels with appropriate interface labels.

The key-expansion steps yield the following constructive statement:

Lemma 6. *The protocol* $(\mathsf{expc13},\mathsf{exps13})$ *constructs the parallel composition of keys* $\bigotimes_{(I,J)\in\mathcal{P}}\llbracket = =\!\!\bullet_{n\,\mathsf{cphs},n}\rrbracket^{(I,J)}$ *from the secret key resource* DHKEY*, for* $\mathcal{P} \coloneqq \{(C,S/\rho(C)) : C\in\mathcal{C}\}\cup\{(E/\eta,S/\eta) : \eta\notin\rho(\mathcal{C})\}$*. The construction holds under the assumptions that HKDF is a KDF with seed* 0*, and that HKDF expansion and HMAC are PRFs. In more detail, for the simulator* σ *and the reductions* $\mathbf{C}_1,\ldots,\mathbf{C}_5$ *described in the proof,*

$$\mathsf{DHKEY}_{\rho,AUX,n} \quad \xmapsto{(\mathsf{expc13},\mathsf{exps13}),\sigma,(\varepsilon,\varepsilon')} \quad \bigotimes_{(I,J)\in\mathcal{P}} \llbracket = =\!\!\bullet_n\rrbracket^{(I,J)}$$

where, for all distinguishers $\mathbf{D}$*,* $\varepsilon'(\mathbf{D}) = n\cdot\varepsilon(\mathbf{D})$ *and*

$$\varepsilon(\mathbf{D}) = |\mathcal{C}|\cdot\big(\varepsilon_{\mathsf{kdf}}(\mathbf{DC}_1)+\varepsilon_{\mathsf{prf}}(\mathbf{DC}_2)+\varepsilon_{\mathsf{kdf}}(\mathbf{DC}_3)+\varepsilon_{\mathsf{prf}}(\mathbf{DC}_4)+\varepsilon_{\mathsf{hmac}}(\mathbf{DC}_5)\big).$$

The Record Layer. The authenticated key resource $= =\!\!\bullet_n$ constructed in the previous step yields sets of keys (*htk*, *atk*, *fsk*) and the finished messages. The gap between the resource $= =\!\!\bullet_n$ and our goal resource, i.e., the unilaterally-secure channel $\twoheadleftarrow \twoheadrightarrow\!\!\bullet_n$, is bridged by a pair of converters essentially exchanging and verifying the finished messages, then using authenticated encryption to protect messages. The key property of our constructed resource, $\twoheadleftarrow \twoheadrightarrow\!\!\bullet_n$, is notably that it allows for messages to be securely (confidentially and authentically) transmitted, either consistently between the server and the honest client, or between the server and the adversary (but never between the client and the adversary).

For TLS 1.3 the record-layer protocol is specified based on authenticated encryption with associated data (AEAD). This mode has been analyzed by Badertscher et al. [1] in recent work. Their result can be "imported" into our work. Thus, for the final step of the proof, we rely on the security of AEAD encryption, which is defined in terms of indistinguishability between two systems $\mathbf{G}_0^{\mathsf{aead}}$ and $\mathbf{G}_1^{\mathsf{aead}}$, formally detailed in the full version. In $\mathbf{G}_0^{\mathsf{aead}}$, encryption and decryption queries to the scheme are answered by encryption and decryption using the given nonce and associated data. For $\mathbf{G}_1^{\mathsf{aead}}$, encryption queries are answered with uniformly random strings of appropriate length, while decryption queries are answered either with a corresponding plaintext (if they were output by a previous encryption query) or by a special *invalid* symbol otherwise.

Lemma 7. *The protocol* $(\mathsf{aeadc},\mathsf{aeads})$ *constructs from the authenticated key resource* $= =\!\!\bullet_n$ *the unilaterally secure channel* $\twoheadleftarrow \twoheadrightarrow\!\!\bullet_n$*, under the assumption that the underlying AEAD cipher is secure. More formally, for the simulator* σ *and the reduction* $\mathbf{C}$ *described in the proof,*

$$= =\!\!\bullet_n \quad \xmapsto{(\mathsf{aeadc},\mathsf{aeads}),\sigma,(0,\varepsilon)} \quad \twoheadleftarrow \twoheadrightarrow\!\!\bullet_n,$$

with $\varepsilon(\mathbf{D}) \coloneqq 2\cdot\varepsilon_{\mathsf{aead}}(\mathbf{C})$ *for all distinguishers* $\mathbf{D}$*.*

Re-constructing TLS. At this point, using the composition theorem completes the proof of Theorem 2. In the full version, we also explain in detail how the composition of all the converters from the modular-steps yields the TLS protocol.

5 Composition with Password-Based Authentication

In prior work, Maurer et al. [22, 24] have discussed means of authenticating a unilaterally authenticated key by using password-based authentication. Thus, by starting from a unilateral key resource (similar to our $= \;=\!\!\bullet_n$ resource), one can use a password — a key with relatively low entropy — shared between a client and a server to obtain a key for which both client and server have authenticity guarantees, and which is sometimes denoted as $\bullet\!\!= \;=\!\!\bullet$ (the bullet on the left hand side indicates that the client is also authenticated). The resources $= \;=\!\!\bullet_n$ and $\bullet\!\!= \;=\!\!\bullet$ are different in that in $= \;=\!\!\bullet_n$ the attacker at the E-interface can also inject a key to be shared with the server (no client authentication). For $\bullet\!\!= \;=\!\!\bullet$ this is no longer possible.

We use the same ideas here, but our goal is to construct the fully secure channel $\bullet\!\twoheadleftarrow \;\twoheadrightarrow\!\bullet$ described below from the unilaterally secure bidirectional $\twoheadleftarrow \;\twoheadrightarrow\!\bullet_n$ and a password.

$\bullet\!\twoheadleftarrow \;\twoheadrightarrow\!\bullet$

No attacker present: Behave as a (multi-message) channel between interfaces C and S.
Attacker present: Provide a secure multiple-use (i.e., keep a buffer of undelivered messages) channel between C and S. In particular:

- On input a message $m \in \mathcal{PT}$ at the C-interface, output $|m|$ at interface E.
- On input (`deliver`, `client`) at the E-interface, deliver the next message at S.
- On input a message $m \in \mathcal{PT}$ at the S-interface, output $|m|$ at interface E.
- On input (`deliver`, `server`) at the E-interface, deliver the next message at C.

The protocol consists of two simple converters: sending the password (client) and verifying it (server), abbreviated as $\mathsf{pwd} = (\mathsf{pwd.send}, \mathsf{pwd.check})$. After the password exchange, the converters simply send and receive messages via the channel. For simplicity, we assume that the server accepts the same user password only once; this can be generalized along the lines of [24, Theorem 4.17]. We model a password distribution with maximum guessing probability ϵ as an additional resource $\mathbf{Q}$. The constructive statement we postulate is:

Lemma 3. *Sending and checking a password constructs from* $\twoheadleftarrow \;\twoheadrightarrow\!\bullet_n$ *the channel* $\bullet\!\twoheadleftarrow \;\twoheadrightarrow\!\bullet$, *for a distribution* $\mathbf{Q}$ *of passwords as described above. More formally, there is a simulator* σ *such that,*

$$[\twoheadleftarrow \;\twoheadrightarrow\!\bullet_n, \mathbf{Q}] \quad \overset{\mathsf{pwd},\sigma,(0,\epsilon)}{\Longmapsto} \quad \bullet\!\twoheadleftarrow \;\twoheadrightarrow\!\bullet.$$

Proof (Sketch). The availability condition follows since the client and the server obtain the same password. The simulator works as follows:

- the session between the honest client and the server is handled by (essentially) forwarding the communication between the E-interface of the constructed resource and the distinguisher,
- for all other sessions, the simulator simply drops all messages provided at its outside interface.

The only way for the distinguisher to be successful in distinguishing between the two cases is by guessing the correct password, since otherwise the behavior is the same in both cases. Since the server accepts a password only once, we can bound the overall success probability of the distinguisher by ϵ. □

Acknowledgments. Ueli Maurer was supported by the Swiss National Science Foundation (SNF), project no. 200020-132794. Björn Tackmann was supported by the Swiss National Science Foundation (SNF) via Fellowship no. P2EZP2_155566 and the NSF grants CNS-1228890 and CNS-1116800. Daniele Venturi acknowledges support by the European Commission (Directorate General Home Affairs) under the GAINS project HOME/2013/CIPS/AG/4000005057, and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 644666.

References

1. Badertscher, C., Matt, C., Maurer, U., Rogaway, P., Tackmann, B.: Augmented secure channels as the goal of the TLS record layer. In: Au, M.H., Miyaji, A. (eds.) Provable Security. LNCS, vol. 9451. Springer, Heidelberg (2015)
2. Bellare, M., Kohno, T., Namprempre, C.: Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. ACM Trans. Inf. Syst. Secur. (TISSEC) **7**(2), 206–241 (2004)
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
4. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
5. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P.Y.: Triple handshakes and cookie cutters: breaking and fixing authentication over TLS. In: IEEE Symposium on Security and Privacy (SP'14). IEEE (2014)
6. Brzuska, C., Fischlin, M., Smart, N., Warinschi, B., Williams, S.: Less is more: relaxed yet composable security notions for key exchange. Int. J. Inf. Secur. **12**(4), 267–297 (2013)
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, July 2013
8. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
9. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 337–351. Springer, Heidelberg (2002)

10. Krawczyk, H.: HMQV: a high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
11. Canetti, R., Shahaf, D., Vald, M.: Universally composable authentication and key-exchange with global PKI. Cryptology ePrint Archive Report 2014/432, October 2014
12. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.2. RFC 5246, August 2008. http://www.ietf.org/rfc/rfc5246.txt
13. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.3. RFC draft, April 2015. http://tlswg.github.io/tls13-spec/
14. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: ACM Conference on Computer and Communications Security 2015 (2015)
15. Hickman, K.: The SSL protocol, February 1995. https://tools.ietf.org/html/draft-hickman-netscape-ssl-00 (internet draft)
16. Jost, D.: A Constructive Analysis of IPSec. Master's thesis, ETH Zürich, April 2014
17. Kohlweiss, M., Maurer, U., Onete, C., Tackmann, B., Venturi, D.: (De-)constructing TLS. Cryptology ePrint Archive, Report 020/2014 (2014)
18. Krawczyk, H.: Cryptographic extraction and key derivation: the HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg (2010)
19. Krawczyk, H., Wee, H.: The OPTLS protocol and TLS 1.3. Manuscript, September 2015
20. Maurer, U.: Constructive cryptography – a new paradigm for security definitions and proofs. In: Mödersheim, S., Palamidessi, C. (eds.) TOSCA 2011. LNCS, vol. 6993, pp. 33–56. Springer, Heidelberg (2012)
21. Maurer, U., Renner, R.: Abstract cryptography. In: Innovations in Computer Science. Tsinghua University Press (2011)
22. Maurer, U., Tackmann, B., Coretti, S.: Key exchange with unilateral authentication: Composable security definition and modular protocol design. Cryptology ePrint Archive, Report 2013/555 (2013)
23. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, pp. 184–200. IEEE (2001)
24. Tackmann, B.: A Theory of Secure Communication. Ph.D. thesis, ETH Zürich (2014)