J Cryptol (2021)34:30
https://doi.org/10.1007/s00145-021-09388-x

Journal of
CRYPTOLOGY

On the Tight Security of TLS 1.3: Theoretically Sound Cryptographic Parameters for Real-World Deployments*

Denis Diemert · Tibor Jager
University of Wuppertal, Wuppertal, Germany
denis.diemert@uni-wuppertal.de
tibor.jager@uni-wuppertal.de

Communicated by Colin Boyd.

Received 31 October 2019 / Revised 17 July 2020 / Accepted 24 July 2020
Online publication 4 June 2021

Abstract. We consider the *theoretically sound* selection of cryptographic parameters, such as the size of algebraic groups or RSA keys, for TLS 1.3 in practice. While prior works gave security proofs for TLS 1.3, their security loss is *quadratic* in the total number of sessions across all users, which due to the pervasive use of TLS is huge. Therefore, in order to deploy TLS 1.3 in a theoretically sound way, it would be necessary to compensate this loss with unreasonably large parameters that would be infeasible for practical use at large scale. Hence, while these previous works show that in principle the design of TLS 1.3 is secure in an asymptotic sense, they do not yet provide any useful *concrete* security guarantees for real-world parameters used in practice. In this work, we provide a new security proof for the cryptographic core of TLS 1.3 in the random oracle model, which reduces the security of TLS 1.3 *tightly* (that is, with constant security loss) to the (multi-user) security of its building blocks. For some building blocks, such as the symmetric record layer encryption scheme, we can then rely on prior work to establish tight security. For others, such as the RSA-PSS digital signature scheme currently used in TLS 1.3, we obtain at least a *linear* loss in the number of users, independent of the number of sessions, which is much easier to compensate with reasonable parameters. Our work also shows that by replacing the RSA-PSS scheme with a tightly secure scheme (e.g., in a future TLS version), one can obtain the first fully tightly secure TLS protocol. Our results enable a theoretically sound selection of parameters for TLS 1.3, even in large-scale settings with many users and sessions per user.

Keywords. Transport Layer Security (TLS), Tightness, Provable security, Key exchange, Cryptographic protocols.

*Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, Grant Agreement 802823.

1. Introduction

Provable security and tightness In modern cryptography, a formal security proof is often considered a minimal requirement for newly proposed cryptographic constructions. This holds in particular for rather complex primitives, such as authenticated key exchange protocols like TLS. The most recent version of this protocol, TLS 1.3, is the first to be developed according to this approach.

A security proof for a cryptographic protocol usually shows that an adversary $\mathcal{A}$ on the protocol can be efficiently converted into an adversary $\mathcal{B}$ solving some conjectured-to-be-hard computational problem. More precisely, the proof would show that any adversary $\mathcal{A}$ running in time $t_{\mathcal{A}}$ and having advantage $\epsilon_{\mathcal{A}}$ in breaking the protocol implies an adversary $\mathcal{B}$ with running time $t_{\mathcal{B}}$ and advantage $\epsilon_{\mathcal{B}}$ in breaking the considered computational problem, such that

$$\frac{\epsilon_{\mathcal{A}}}{t_{\mathcal{A}}} \leq \ell \cdot \frac{\epsilon_{\mathcal{B}}}{t_{\mathcal{B}}} \tag{1}$$

where ℓ is bounded.[1] Following the approach of Bellare and Ristenpart [9,10] to measure concrete security, the terms $\epsilon_{\mathcal{A}}/t_{\mathcal{A}}$ and $\epsilon_{\mathcal{B}}/t_{\mathcal{B}}$ are called the *"work factors"*[2] of adversaries $\mathcal{A}$ and $\mathcal{B}$, respectively, and the factor ℓ is called the *"security loss"* of the reduction. We say that a security proof is "tight", if ℓ is small (e.g., constant).

Concrete security In classical complexity-theoretic cryptography, it is considered sufficient if ℓ is *asymptotically* bounded by a polynomial in the security parameter. However, the *concrete* security guarantees that we obtain from the proof depend on the *concrete* loss ℓ of the reduction and (conjectured) *concrete* bounds on $\epsilon_{\mathcal{B}}/t_{\mathcal{B}}$. Thus, in order to obtain meaningful results for the concrete security of cryptosystems, we need to be more precise and make these quantities explicit.

If for a given protocol we have an concrete upper bound $\epsilon_{\mathcal{A}}/t_{\mathcal{A}}$ on the work factor of any adversary $\mathcal{A}$, then we can say that the protocol provides "security equivalent to $-\log_2(\epsilon_{\mathcal{A}}/t_{\mathcal{A}})$ bits". However, note that these security guarantees depend on the loss ℓ of the reduction and a bound on $\epsilon_{\mathcal{B}}/t_{\mathcal{B}}$. More concretely, suppose that we aim for a security level of, say, "128-bit security". That is, we want to achieve $-\log_2(\epsilon_{\mathcal{A}}/t_{\mathcal{A}}) \geq 128$. A security proof providing (1) with some concrete security loss ℓ would allow us to achieve this via

$$-\log_2(\epsilon_{\mathcal{A}}/t_{\mathcal{A}}) \geq -\log_2(\ell \cdot \epsilon_{\mathcal{B}}/t_{\mathcal{B}}) \geq 128$$

To this end, we have to make sure that it is reasonable to assume that $\epsilon_{\mathcal{B}}/t_{\mathcal{B}}$ is small enough, such that $-\log_2(\ell \cdot \epsilon_{\mathcal{B}}/t_{\mathcal{B}}) \geq 128$. Indeed, we can achieve this by choosing

[1] The exact bound on ℓ depends on the setting. For instance, in the asymptotic setting, as described below, ℓ is bounded by a polynomial.

[2] Opposed to Bellare and Ristenpart, we consider the inverse of their work factor just to avoid dividing by 0 in the somewhat artificial case in which $\epsilon = 0$. We may assume that $t > 0$ as the adversary at least needs to read its input. This does not change anything other than we need to consider the negative logarithm for the bit security level.

cryptographic parameters (such as Diffie–Hellman groups or RSA keys) such that indeed it is reasonable to assume that $\epsilon_{\mathcal{B}}/t_{\mathcal{B}}$ is sufficiently small. Hence, by making the quantities ℓ and $\epsilon_{\mathcal{B}}/t_{\mathcal{B}}$ explicit, the concrete security approach enables us to choose cryptographic parameters in a *theoretically sound* way, such that $\epsilon_{\mathcal{B}}/t_{\mathcal{B}}$ is sufficiently small and thus we *provably* achieve our desired security level.

However, note that if the security loss ℓ is "large", then we need to compensate this with a "smaller" $\epsilon_{\mathcal{B}}/t_{\mathcal{B}}$. Of course, we can easily achieve this by simply choosing the cryptographic parameters large enough, but this might significantly impact the computational efficiency of the protocol. In contrast, if the security proof is "tight", then ℓ is "small" and we can accordingly use smaller parameters, while still being able to instantiate and deploy our protocol in a theoretically sound way.

Since our focus is on the proof technique for TLS 1.3, we chose to consider this simple view on bit security. Alternatively, Micciancio and Walter [58] recently proposed a formal notion for bit security. They try to overcome paradoxical situations occurring with a simple notion of bit security as discussed above. The paradox there is that sometimes the best possible advantage is actually higher than the advantage against an idealized primitive, which is usually considered for bit security. As an example, they mention pseudorandom generators (PRG) for which it was shown that the best possible attack in distinguishing the PRG from random using an n-bit seed value has advantage $2^{-n/2}$ [30] (i.e., $n/2$ bits of security), even though the best seed recovery attack (with advantage 2^{-n}) does not contradict n-bit security. However, these paradoxical situations mostly occur in the non-uniform setting, in which the adversary receives additional information and thus allows the adversary to gain higher advantages. As the discussion above should serve only for motivation and we do not consider non-uniform adversaries, we believe that the simple, intuitive view on bit security we chose here is sufficient.

Theoretically sound deployment of TLS Due to the lack of tight security proofs for TLS 1.3, we are currently not able to deploy TLS 1.3 in a theoretically sound way with reasonable cryptographic parameters. All current security proofs for different draft versions of TLS 1.3 [31–33,37,40] have a loss $\ell \geq n_s^2$ which is at least *quadratic* in the total number n_s of sessions.

Let us illustrate the practical impact of this security loss. Suppose we want to choose parameters in a theoretically sound way, based on a security proof with this quadratic loss in the total number of sessions. Given that TLS will potentially be used by billions of systems, each running thousands of TLS sessions over time, it seems reasonable to assume at least 2^{30} users and 2^{15} sessions per user. In this case, we would have $n_s \geq 2^{45}$ sessions over the life time of TLS 1.3. This yields a security loss of $\ell \geq n_s^2 = 2^{90}$, i.e., we lose "90 bits of security".

Choosing practical parameters If we now instantiate TLS with parameters that provide "128-bit security" (more precisely, such that it is reasonable to assume that $-\log_2(\epsilon_{\mathcal{B}}/t_{\mathcal{B}}) = 128$ for the best possible adversary $\mathcal{B}$ on the underlying computational problem), then the existing security proofs guarantee only $128 - 90 = 38$ "bits of security" for TLS 1.3, which is very significantly below the desired 128 bits.

Hence, from a *concrete security* perspective, the current proofs are not very meaningful for typical cryptographic parameters used in practice today.

Choosing theoretically sound parameters If we want to *provably* achieve "128-bit security" for TLS 1.3, we would need to deploy the protocol with cryptographic parameters that compensate the 90-bit security loss. Concretely, this would mean that an elliptic curve Diffie–Hellman group of order $\approx 2^{256}$ must be replaced with a group of order at least $\approx 2^{436}$. The impact on RSA keys, as commonly used for digital signatures in TLS 1.3, is even more significant. While a modulus size of 3072 bits is today considered sufficient to provide "128-bit security", a modulus size of more than 10,000 bits would be necessary to compensate the 90-bit security loss.[3]

For illustration, consider ECDSA instantiated with NIST P-256 and instantiated with NIST P-384 (resp. NIST P-521), which are the closest standard curves to the calculated group order 2^{436} to compensate 90-bit security loss. The `openssl speed` benchmark shows that this would result in significantly decreasing the number of both signature computations and signature verifications per second. Concretely, for NIST P-256 we obtain $\approx$39,407 signature computations per second and $\approx$14,249 signature verifications per seconds, whereas replacing NIST P-256 by the next larger NIST P-384 (resp. NIST P-521) we only obtain $\approx$1102 (resp. $\approx$3437) signature computations per second and $\approx$1479 (resp. $\approx$1715) signature verifications per seconds. For RSA, we measured for a modulus size of 3072 bits, $\approx$419 signature computations per second and $\approx$20,074 signature verifications per seconds, and for a modulus size of 15, 360 bits, $\approx$4 signature computations per second and 880 signature verifications per second.[4]

Due to the significant performance penalty of these increased parameters, it seems impractical for most applications to choose parameters in a theoretically sound way. This includes both "large-scale" TLS deployments, e.g., at content distribution providers or major Web sites, for which this would incur significant additional costs, as well as "small-scale" deployments, e.g., in Internet-of-Things applications with resource-constrained devices.

In practice, usually the first approach is followed, due to the inefficiency of the theoretically sound approach. However, we believe it is a very desirable goal to make it possible to follow the theoretically sound approach in practice, by giving improved, tighter security proofs. This is the main motivation behind the present paper.

Our contributions and approach We give the first tight security proof for TLS 1.3, and thereby the first tight security proof for a real-world authenticated key exchange protocol used in practice. The proof covers both mutual and server-only authentication.

[3] Cf. https://www.keylength.com/ and the various documents by different standardization bodies referenced there.

[4] Generated on a Apple MacBook Pro (13-inch, 2019, Four Thunderbolt 3 ports) running macOS 10.15.3 and OpenSSL 1.1.1d (10 Sep 2019) on a 2,4 GHz Quad-Core Intel Core i5 (Coffee Lake, 8279U) CPU with 16 GB (2133 MHz LPDDR3) RAM.

The former setting is commonly considered in cryptographic research, but the latter is much more frequently used in practice.

Our proof reduces the security of TLS to appropriate *multi-user* security definitions for the underlying building blocks of TLS 1.3, such as the digital signature scheme, the HMAC and HKDF functions, and the symmetric encryption scheme of the record layer. Further, the proof is under the strong Diffie–Hellman (SDH) [1] assumption in the random oracle model. In contrast, standard-model proofs often require a PRF-ODH-like assumption [43]. However, these assumptions are closely related. Namely, as shown by Brendel et al. [21], PRF-ODH is implied by SDH in the random oracle model (see also [21] for an analysis of various variants of the PRF-ODH assumption). One technical contribution of our work is the observation that using the same two assumptions explicitly in the security proof in combination with modeling the key derivation of TLS 1.3 as multiple random oracles [11], we obtain leverage for a *tight* security proof. For details on how we use this see below.

Another technical contribution of our work is to identify and define reasonable multi-user definitions for these building blocks, and to show that these are sufficient to yield a tight security proof. These new definitions make it possible to independently analyze the multi-user security of the building blocks of TLS 1.3.

These building blocks can be instantiated as follows.

Symmetric encryption Regarding the symmetric encryption scheme used in TLS 1.3, we can rely on previous work by [13] and [41], who gave tight security proofs for the AES-GCM scheme and also considered the nonce-randomization mechanism adopted in TLS 1.3.

HMAC and HKDF For the HMAC and HKDF functions, which are used in TLS 1.3 to perform message authentication and key derivation, we give new proofs of tight multi-user security in the random oracle model.

Signature schemes TLS 1.3 specifies four signature schemes, RSA-PSS [26,59], RSA-PKCS #1 v1.5 [48,59], ECDSA [45], and EdDSA [14,46]. Due to the fact that RSA-based public keys are most common in practice, the RSA-based schemes currently have the greatest practical relevance in the context of TLS 1.3.

Like previous works on tightly secure authenticated key exchange [4,38], we require *existential unforgeability in the multi-user setting with adaptive corruptions*. Here, two dimensions are relevant for tightness: (i) the number of signatures issued per user, and (ii) the number of users.

- RSA-PSS is the recommended signature scheme in TLS 1.3. It has a tight security proof in the number of signatures per user [26,47], but not in the number of users.
- RSA-PKCS #1 v1.5 also has a tight security proof [42] in the number of signatures per user, but not in the number of users. However, we note that this proof requires to double the size of the modulus, and also that it requires a hash function with "long" output (about half of the size of the modulus), and therefore does not immediately apply to TLS 1.3.
- For ECDSA there exists a security proof [35] that considers a weaker "one-signature-per-message" security experiment. While this would be sufficient for our result (because the signatures are computed over random nonces which most likely are unique), their security proof is not tight.

We discuss the issue of non-tightness in the number of users below.

In contrast to previously published security proofs, which considered preliminary drafts of TLS 1.3, we consider the final version of TLS 1.3, as specified in RFC 8446. However, the differences are minor, and we believe that the published proofs for TLS 1.3 drafts also apply to the final version without any significant changes. We first focus on giving a tight security proof for the TLS 1.3 handshake. Then, following [40] we show how to generically compose the handshake with a symmetric encryption scheme to obtain security of the full protocol. Since we focus on efficiency of practical deployments, our security proof of TLS 1.3 is in the random oracle model [11].

Features of TLS omitted in the security analysis As common in previous cryptographic security analyses of the TLS protocol [31,33,40,43,55], we consider the *"cryptographic core"* of TLS 1.3. That is, our analysis only focuses on the TLS 1.3 Full 1-RTT (EC)DHE Handshake and its composition with an arbitrary symmetric key protocol. The full TLS 1.3 standard allows the *negotiation* of different ciphersuites (i.e., AEAD algorithm and hash algorithm), DH groups, and signature algorithms, but this negotiation is out of scope of our work and we focus on a fixed selection of algorithms. Similarly, we do not consider version negotiation and backward compatibility as, e.g., considered in [17,34]. Instead, we only focus on clients and servers that negotiate TLS 1.3. We also do not consider advanced, optional protocol features, such as abbreviated session resumption based on pre-shared keys (PSK) (with optional (EC)DHE key exchange and 0-RTT, as in e.g., [31,33]). That is, we consider neither PSKs established using TLS nor PSKs established using some out-of-band mechanism. Further, we ignore the TLS 1.3 record layer protocol, which performs transmission of cryptographic messages (handshake messages and encrypted data) on top of the TCP protocol and below the cryptographic protocols used in TLS. Additionally, we omit the alert protocol [65, Sect. 6] and the considerations of extensions, such as post-handshake client authentication [54]. Furthermore, we do not consider ciphersuite downgrade or protocol version rollback attacks as discussed in [44,57,69]. Hence, we abstract the cryptographic core of TLS in essentially the same way as in [31,33,40,43,55]. See for instance [19,28] for a different approach, which analyses a concrete reference implementation of TLS (miTLS) with automated verification tools.

However, as mentioned earlier, we discuss the composition of the TLS 1.3 Full (EC)DHE Handshake with the nonce randomization mechanism of AES-GCM, which could be proven to be tightly secure by Hoang et al. [41] and is a first step toward a tight composition with the actual record protocol.

Achieving tightness using the random oracle model Conceptually, we adopt a technique of Cohn-Gordon et al. [25] to TLS 1.3. The basic idea of the approach is that the random oracle and random self-reducibility of SDH allow us to embed a single SDH challenge into every protocol session simultaneously. The DDH oracle provided by the SDH experiment allows us to guarantee that we are able to recognize a random oracle query that corresponds to a solution of the given SDH instance without tightness loss. A remarkable difference to [25] is that they achieve only a linear tightness loss in the number of users, and show to be optimal for the class of high-efficiency protocols considered there. Previous proofs for different TLS versions suffered from the general difficulty of

proving tight security of AKE protocols, such as the "commitment problem" described in [38]. We show that the design of TLS 1.3 allows a tightly secure proof with constant security loss.

Relation to previous non-tight security proofs in the standard model We stress that our result is not a strict improvement over previous security proofs for TLS 1.3 [31,33,40,43,55], in particular not to standard model proofs without random oracles. Rather, our objective is to understand under which exact assumptions a tight security proof, and thus a theoretically sound instantiation with optimal parameters such as group sizes is possible. We show that the random oracle model allows this. Hence, if one is willing to accept the random oracle model as a reasonable heuristic, then one can use optimal parameters. Otherwise, either no theoretically sound deployment is (currently) possible, or larger parameters must be used to overcome the loss.

Tight security of signature schemes in the number of users All signature schemes in TLS have in common that they currently do not have a tight security proof in the number of users. Since all these schemes have unique secret keys in the sense of [5], Bader et al. even showed that they cannot have a tight security proof, at least not with respect to what they called a "simple" reduction.

There are several ways around this issue:

1. We can compensate the loss by choosing larger RSA keys. Note that the security loss is only *linear* in the number of *users*. For instance, considering 2^{30} users as above, we would lose only "30 bits of security". This might be compensated already with a 4096-bit RSA key, which is already quite common today.
   Most importantly, due to our modular security proof, this security loss impacts *only* the signature keys. In contrast, for previous security proofs one would have to increase *all* cryptographic parameters accordingly (or require a new proof).
2. Alternatively, since the RSA moduli in the public keys of RSA-based signature schemes are independently generated, they do not share any common parameters, such as a common algebraic group as for many tightly secure Diffie–Hellman-based schemes. On the one hand, this makes a tight security proof very difficult, because there is no common algebraic structure that would allow for, e.g., random self-reducibility. The latter is often used to prove tight security for Diffie–Hellman-based schemes. On the other hand, one can also view this as a security advantage. The same reason that makes it difficult for us to give a tight security proof in the number of users, namely that there is no common algebraic structure, seems also to make it difficult for an adversary to leverage the availability of more users to perform a more efficient attack than on a single user. Hence, it seems reasonable to assume that tightness in the number of users is not particularly relevant for RSA-based schemes, and therefore we do not have to compensate any security loss. This is an additional assumption, but it would even make it possible to choose *optimal* parameters, independent of the number of users.
3. Finally, in future revisions of TLS one could include another signature scheme which is tightly secure in both dimensions, such as the efficient scheme recently constructed by [38].

Further related work The design of TLS 1.3 is based on the OPTLS protocol by [56], which, however, does not have a tight security proof.

Constructing tightly secure authenticated key exchange protocols has turned out to be a difficult task. The first tightly secure AKE protocols were proposed by [4]. Their constructions do not have practical efficiency and are therefore rather theoretical. Notably, they achieve proofs in the standard model, that is, without random oracles or similar idealizations.

Recently, [38] published the first practical and tightly secure AKE protocol. Their protocol is a three-round variant of the signed Diffie–Hellman protocol, where the additional message is necessary to avoid what is called the "commitment problem" in [38]. Our result also shows implicitly that TLS is "out-of-the-box" able to avoid the commitment problem, without requiring an additional message. Furthermore, [38] describe an efficient digital signature scheme with tight security in the multi-user setting with adaptive corruptions. As already mentioned above, this scheme could also be used in TLS 1.3 in order to achieve a fully tight construction.

[25] constructed extremely efficient AKE protocols, but with security loss that is linear in the number of users. They also showed that this linear loss is unavoidable for many types of protocols.

Formal security proofs for (slightly modified variants of) prior TLS versions were given, e.g., in [15,16,19,22,43,55,60].

Concurrent and independent work In concurrent and independent work, Davis and Günther [27] studied the tight security of the SIGMA protocol [51] and the main TLS 1.3 handshake protocol. Similar to our proof (see Theorem 6) they reduce the security of the TLS 1.3 handshake in the random oracle to the hardness of strong DH assumption (SDH), the collision resistance of the hash function, and the multi-user security of the signature scheme and the PRFs. However, we would like to point out that there are some notable differences between their work and ours:

- We use the multi-stage key exchange model from [36], which allows us to show security for all intermediate, internal keys and further secrets derived during the handshake. They use a code-based authenticated key exchange model, which considers mutual authentication and the negotiation of a single key, namely the final session key that is used in the TLS 1.3 record layer.
- Our work makes slightly more extensive use of the random oracle model. Concretely, both security proofs need to deal with the fact that the TLS 1.3 key derivation does not bind the DH key to the context used to derive a key in a single function. We resolve this by modeling several functions as random oracles, while Davis and Günther [27] model the functions HKDF.Extract and HKDF.Expand of the HKDF directly as random oracles and are able to circumvent the above problem by using efficient book-keeping in the proof.
- Since the multi-stage key exchange model [36] provides a tightly secure composition theorem, we were able to make a first step toward a tight security proof for the composition of the TLS handshake with the TLS record layer by leveraging known security proofs for AES-GCM by [13] and [41].

– Davis and Günther [27] focused only on the tight security of the handshake protocol of TLS 1.3, but provide an extensive evaluation of the concrete security implications of their bounds when instantiated with various amounts of resources. Furthermore, they even give a bound for the strong DH assumption in the generic group model (GGM) and were able to show that SDH is as hard as the discrete logarithm problem in the GGM.

Hence, neither of these two independent works covers the other, both papers make complementary contributions toward understanding the theoretically sound deployment of TLS in practice.

Future work and open problems A notable innovative feature of TLS 1.3 is its 0-RTT mode for low-latency key exchange, which we do not consider in this work. We believe it is an interesting open question to analyze whether tight security can also achieved for the 0-RTT mode. Probably along with full forward security, as considered in [2].

Furthermore, we consider TLS 1.3 "in isolation", that is, independent of other protocol versions that may be provided by a server in parallel in order to maximize compatibility. It is known that this might yields cross-protocol attacks, such as those described in [3,18,44,57]. It would be interesting to see whether (tight) security can also be proven in a model that considers such backwards compatibility issues as, e.g., in [17,34], and which exact impact on tightness this would have, if any. A major challenge in this context is to tame the complexity of the security model and the security proof.

2. Preliminaries

In this section, we introduce notation used in this paper and recall definitions of fundamental building blocks as well as their corresponding security models.

2.1. *Notation*

We denote the empty string, i.e., the string of length 0, by ε. For strings a and b, we denote the concatenation of these strings by $a \parallel b$. For an integer $n \in \mathbb{N}$, we denote the set of integers ranging from 1 to n by $[n] := \{1, \ldots, n\}$. For a set $X = \{x_1, x_2, \ldots\}$, we use $(v_i)_{i \in X}$ as a shorthand for the tuple $(v_{x_1}, v_{x_2}, \ldots)$. We denote the operation of assigning a value y to a variable x by $x := y$. If S is a finite set, we denote by $x \xleftarrow{\$} S$ the operation of sampling a value uniformly at random from set S and assigning it to variable x. If $\mathcal{A}$ is an algorithm, we write $x := \mathcal{A}(y_1, y_2, \ldots)$, in case $\mathcal{A}$ is deterministic, to denote that $\mathcal{A}$ on inputs $y_1, y_2, \ldots$ outputs x. In case $\mathcal{A}$ is probabilistic, we overload notation and write $x \xleftarrow{\$} \mathcal{A}(y_1, y_2, \ldots)$ to denote that random variable x takes on the value of algorithm $\mathcal{A}$ ran on inputs $y_1, y_2, \ldots$ with fresh random coins. Sometimes we also denote this random variable simply by $\mathcal{A}(y_1, y_2, \ldots)$.

2.2. *Advantage Definitions Versus Security Definitions*

Due to the real-world focus of this paper, we follow the *human-ignorance approach* proposed by Rogaway [66,67] for our security definitions and statements. As a consequence, we drop security parameters in all of our syntactical definitions. This way we reflect the algorithms as they are used in practice more appropriately. The human-ignorance approach also allows us, e.g., to consider a fixed group opposed to the widely used approach of employing a group generator in the asymptotic security setting. We believe that doing so brings us closer to the actual real-world deployment of the schemes. In terms of wording, we can never refer to any scheme as being "secure" in a formal context. Formally, we only talk about advantages and success probabilities of adversaries.

2.3. *Diffie–Hellman Assumptions*

We start with the definitions of the standard Diffie–Hellman (DH) assumptions [20,29].

Definition 1. (*Computational Diffie–Hellman Assumption*) Let $\mathbb{G}$ be a cyclic group of prime order q, and let g be a generator of $\mathbb{G}$. We denote the *advantage of an adversary $\mathcal{A}$ against the computational Diffie–Hellman (CDH) assumption* by

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{A}) := \Pr[a, b \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(g^a, g^b) = g^{ab}].$$

Definition 2. (*Decisional Diffie–Hellman Assumption*) Let $\mathbb{G}$ be a cyclic group of prime order q and let g be a generator of $\mathbb{G}$. We denote the *advantage of an adversary $\mathcal{A}$ against the decisional Diffie–Hellman (DDH) assumption* by

$$\begin{aligned}\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{A}) := |&\Pr[a, b \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(g^a, g^b, g^{ab}) = 1] \\ &- \Pr[a, b, c \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(g^a, g^b, g^c) = 1]|.\end{aligned}$$

Following [1], we also consider the *strong Diffie–Hellman (SDH)* assumption. The SDH problem is essentially the CDH problem except that the adversary has additionally access to a DDH oracle. The DDH oracle outputs 1 on input (g^a, g^b, g^c) if and only if $c = ab \bmod q$. However, we restrict the DDH oracle in the SDH experiment by fixing the first component. Without this restriction, we would consider the *gap Diffie–Hellman* [63] problem.

Definition 3. (*Strong Diffie–Hellman Assumption*) Let $\mathbb{G}$ be a cyclic group of prime order q and let g be a generator of $\mathbb{G}$. Further, let $\mathrm{DDH}(\cdot,\cdot,\cdot)$ denote the oracle that on input $g^a, g^b, g^c \in \mathbb{G}$ outputs 1 if $c = ab \bmod q$ and 0 otherwise. We denote the *advantage of an adversary $\mathcal{A}$ against the strong Diffie–Hellman (SDH) assumption* by

$$\mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{A}) := \Pr[a, b \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}^{\mathrm{DDH}(g^a,\cdot,\cdot)}(g^a, g^b) = g^{ab}].$$

2.4. *Pseudorandom Functions*

Informally, a *pseudorandom function (PRF)* is a keyed function that is indistinguishable from a truly random function. The standard definition only covers the case of a single key (resp. a single user). Bellare et al. introduced the related notion of *multi-oracle families* [8], which essentially formalizes *multi-user security* of a PRF. In contrast to the standard definition, the challenger now implements N oracles instead of a single one. The adversary may ask queries of the form (i, x), which translates to a request of an image of x under the ith oracle. Hence, the adversary essentially plays N "standard PRF experiments" in parallel, except that the oracles all answer either uniformly at random or with the actual PRF.

Definition 4. (MU–PRF-*Security*) Let PRF be an algorithm implementing a deterministic, keyed function $\mathsf{PRF}\colon \mathcal{K}_{\mathsf{PRF}} \times \mathcal{D} \to \mathcal{R}$ with finite key space $\mathcal{K}_{\mathsf{PRF}}$, (possibly infinite) domain $\mathcal{D}$ and finite range $\mathcal{R}$. Consider the following security experiment $\mathsf{Exp}^{\mathsf{MU-PRF}}_{\mathsf{PRF},N}(\mathcal{A})$ played between a challenger and an adversary $\mathcal{A}$:

1. The challenger chooses a bit $b \xleftarrow{\$} \{0, 1\}$, and for every $i \in [N]$ a key $k_i \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$ and a function $f_i \xleftarrow{\$} \{f \mid f\colon \mathcal{D} \to \mathcal{R}\}$ uniformly and independently at random. Further, it prepares a function $\mathcal{O}_b$ such that for $i \in [n]$

$$\mathcal{O}_b(i, \cdot) := \begin{cases} \mathsf{PRF}(k_i, \cdot), & \text{if } b = 0 \\ f_i(\cdot), & \text{otherwise} \end{cases}.$$

2. The adversary may issue queries $(i, x) \in [N] \times \mathcal{D}$ to the challenger adaptively, and the challenger replies with $\mathcal{O}_b(i, x)$.
3. Finally, the adversary outputs a bit $b' \in \{0, 1\}$. The experiment outputs 1 if $b = b'$ and 0 otherwise.

We define the *advantage of an adversary $\mathcal{A}$ against the multi-user pseudorandomness* (MU–PRF) *of* PRF *for N users* to be

$$\mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{PRF},N}(\mathcal{A}) := \left| \Pr[\mathsf{Exp}^{\mathsf{MU-PRF}}_{\mathsf{PRF},N}(\mathcal{A}) = 1] - \frac{1}{2} \right|.$$

where $\mathsf{Exp}^{\mathsf{MU-PRF}}_{\mathsf{PRF},N}(\mathcal{A})$ is defined above.

2.5. *Collision-Resistant Hash Functions*

A *(keyless) hash function* H is a deterministic algorithm implementing a function $\mathsf{H}\colon \mathcal{D} \to \mathcal{R}$ such that usually $|\mathcal{D}|$ is large (possibly infinite) and $|\mathcal{R}|$ is small (finite). Recall the standard notion of *collision resistance* of a hash function.

Definition 5. (*Collision Resistance*) Let H be a keyless hash function. We denote the *advantage of an adversary $\mathcal{A}$ against the collision resistance of* H by

$$\mathsf{Adv}_{\mathsf{H}}^{\mathsf{Coll-Res}}(\mathcal{A}) := \Pr\left[(m_1, m_2) \xleftarrow{\$} \mathcal{A} : m_1 \neq m_2 \wedge \mathsf{H}(m_1) = \mathsf{H}(m_2)\right].$$

2.6. *Digital Signature Schemes*

We recall the standard definition of a *digital signature scheme* by [39].

Definition 6. (*Digital Signature Scheme*) A *digital signature scheme* for message space M is a triple of algorithms $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vrfy})$ such that

1. $\mathsf{SIG.Gen}$ is the randomized key generation algorithm generating a public (verification) key pk and a secret (signing) key sk and takes no input.
2. $\mathsf{SIG.Sign}(sk, m)$ is the randomized signing algorithm outputting a signature σ on input message $m \in M$ and signing key sk.
3. $\mathsf{SIG.Vrfy}(pk, m, \sigma)$ is the deterministic verification algorithm outputting either 0 or 1.

Correctness We say that a digital signature scheme SIG is *correct* if for any $m \in M$, and for any (pk, sk) that can be output by $\mathsf{SIG.Gen}$, it holds

$$\mathsf{SIG.Vrfy}\,(pk, m, \mathsf{SIG.Sign}(sk, m)) = 1.$$

2.6.1. *Existential Unforgeability of Signatures*

The standard notion of security for digital signature schemes is called *existential unforgeability under an adaptive chosen-message attack (EUF-CMA)*. We recall the standard definition [39] next.

Definition 7. ($\mathsf{EUF-CMA}$-*Security*) Let SIG be a digital signature scheme (Definition 6). Consider the following experiment $\mathsf{Exp}_{\mathsf{SIG}}^{\mathsf{EUF-CMA}}(\mathcal{A})$ played between a challenger and an adversary $\mathcal{A}$:

1. The challenger generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{SIG.Gen}$, initializes the set of chosen-message queries $\mathcal{Q}_{\mathsf{Sign}} := \emptyset$, and hands pk to $\mathcal{A}$ as input.
2. The adversary may issue signature queries for messages $m \in M$ to the challenger adaptively. The challenger replies to each query m with a signature $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk, m)$. Each chosen-message query m is added to the set of chosen-message queries $\mathcal{Q}_{\mathsf{Sign}}$.
3. Finally, the adversary outputs a forgery attempt (m^*, σ^*). The challenger checks whether $\mathsf{SIG.Vrfy}(pk, m^*, \sigma^*) = 1$ and $m^* \notin \mathcal{Q}_{\mathsf{Sign}}$. If both conditions hold, the experiment outputs 1 and 0 otherwise.

We denote the *advantage of an adversary $\mathcal{A}$ in breaking the existential unforgeability under an adaptive chosen-message attack* (EUF–CMA) *for* SIG by

$$\mathsf{Adv}^{\mathsf{EUF-CMA}}_{\mathsf{SIG}}(\mathcal{A}) := \Pr\left[\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\mathsf{SIG}}(\mathcal{A}) = 1\right]$$

where $\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\mathsf{SIG}}(\mathcal{A})$ is defined as before.

2.6.2. *Existential Unforgeability of Signatures in a Multi-user Setting*

In a "real-world" scenario, the adversary is more likely faced a different challenge than described in Definition 7. Namely, a real-world adversary presumably plays against multiple users at the same time and might even be able to get the secret keys of a subset of these users. In this setting, its challenge is to forge a signature for any of the users that it has no control of (to exclude trivial attacks). To capture this intuition, we additionally consider the multi-user EUF-CMA notion with adaptive corruptions as proposed by [4].

To that end, the single-user notion given in Definition 7 can naturally be upgraded to a multi-user notion with adaptive corruptions as follows.

Definition 8. ($\mathsf{MU-EUF-CMA}^{\mathsf{corr}}$*-Security*) Let $N \in \mathbb{N}$. Let SIG be a digital signature scheme (Definition 6). Consider the following experiment $\mathsf{Exp}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},N}(\mathcal{A})$ played between a challenger and an adversary $\mathcal{A}$:

1. The challenger generates a key pair $(pk_i, sk_i) \xleftarrow{\$} \mathsf{SIG.Gen}$ for each user $i \in [N]$, initializes the set of corrupted users $\mathcal{Q}^{\mathsf{corr}} := \emptyset$, and N sets of chosen-message queries $\mathcal{Q}_1, \ldots, \mathcal{Q}_N := \emptyset$ issued by the adversary. Subsequently, it hands $(pk_i)_{i\in[N]}$ to $\mathcal{A}$ as input.
2. The adversary may issue signature queries $(i, m) \in [N] \times M$ to the challenger adaptively. The challenger replies to each query with a signature $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk_i, m)$ and adds (m, σ) to $\mathcal{Q}_i$. Moreover, the adversary may issue corrupt queries $i \in [N]$ adaptively. The challenger adds i to $\mathcal{Q}^{\mathsf{corr}}$ and replies sk_i to the adversary. We call each user $i \in \mathcal{Q}^{\mathsf{corr}}$ *corrupted*.
3. Finally, the adversary outputs a tuple (i^*, m^*, σ^*). The challenger checks whether $\mathsf{SIG.Vrfy}(pk_{i^*}, m^*, \sigma^*) = 1$, $i^* \notin \mathcal{Q}^{\mathsf{corr}}$ and $(m^*, \cdot) \notin \mathcal{Q}_{i^*}$. If all of these conditions hold, the experiment outputs 1 and 0 otherwise.

We denote the *advantage of an adversary $\mathcal{A}$ in breaking the multi-user existential unforgeability under an adaptive chosen-message attack with adaptive corruptions* (MU–EUF–CMA) *for* SIG by

$$\mathsf{Adv}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},\,N}(\mathcal{A}) := \Pr\left[\mathsf{Exp}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},\,N}(\mathcal{A}) = 1\right]$$

where $\mathsf{Exp}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},\,N}(\mathcal{A})$ is as defined before.

Remark 1. This notion can also be weakened by excluding adaptive corruptions. The resulting experiment is analogous except that queries to the corruption oracle are forbidden.

The corresponding notions are denoted by MU–EUF–CMA instead of MU–EUF–CMA$^{\text{corr}}$.

2.7. *HMAC*

A prominent deterministic example of a message authentication code (MAC) is HMAC [7, 49]. The construction is based on a cryptographic hash function (Sect. 2.5). As we will model HMAC in the remainder mainly as a PRF (e.g., Sect. 5), we do not formally introduce MACs.

Construction Let H be a cryptographic hash function with output length μ and let be κ be the key-length.

- MAC.Gen: Choose $k \xleftarrow{\$} \{0,1\}^{\kappa}$ and return k.
- MAC.Tag(k, m): Return $t := \mathsf{H}((k \oplus \mathsf{opad}) \parallel \mathsf{H}((k \oplus \mathsf{ipad}) \parallel m))$.
- MAC.Vrfy(k, m, t): Return 1 iff. $t = \mathsf{MAC.Tag}(k, m)$.

where opad and ipad are according to RFC 2104 [49] the bytes `0x5c` and `0x36` repeated B-times, respectively, where B is the block size (in bytes) of the underlying hash function. k is padded with 0's to match the block size B. If k should be larger, then it is hashed down to less and then padded to the right length as before.

2.8. *HKDF Scheme*

The core of the TLS 1.3 key derivation [65, Sect. 7.1] is the *key derivation function (KDF)* HKDF proposed by [52,53] and standardized in RFC 5869 [50]. It follows the *extract-and-expand* [53] paradigm and is based on HMAC (Sect. 2.7). The algorithm consists of two subroutines HKDF.Extract and HKDF.Expand. The function HKDF.Extract is a *randomness extractor* [61,62] that on input a (non-secret and possibly fixed) extractor salt *xts* and a (not necessarily uniformly distributed) source key material *skm* outputs a pseudorandom key *prk*. The function HKDF.Expand is a *variable output length PRF* that on input *prk*, (potentially empty) context information *ctx* and length parameter L outputs a pseudorandom key *km* of length L.

Construction Intuitively, HKDF derives a pseudorandom key (i.e., indistinguishable from a uniformly sampled key) from some source key material and then stretches this pseudorandom key to the desired length. Formally, we have the following construction.

1. $prk := \mathsf{HKDF.Extract}(xts, skm) = \mathsf{HMAC}(xts, skm)$
2. $km = K(1) \parallel \cdots \parallel K(\omega) := \mathsf{HKDF.Expand}(prk, ctx, L)$, where $\omega := \lceil L/\mu \rceil$, μ is the output length of the underlying hash function used in HMAC and $K(i)$ is inductively defined by
   - $K(1) := \mathsf{HMAC}(prk, ctx \parallel 0)$, and
   - $K(i+1) := \mathsf{HMAC}(prk, K(i) \parallel ctx \parallel i)$ for $1 \leq i < \omega$.

   $K(\omega)$ is simply truncated to the first ($L \bmod \mu$) bits to fit the length of L.

We overload notation to denote by $\mathsf{HKDF.Expand}(prk, ctx)$ the function described above for a fixed length parameter L that is clear from the context.

The function HKDF then is just a shorthand for the execution of HKDF.Extract and HKDF.Expand in sequence. That is, on input (xts, skm, ctx, L) it computes $prk := \mathsf{HKDF.Extract}(xts, skm)$ and outputs km with $km := \mathsf{HKDF.Expand}(prk, ctx, L)$.

3. Multi-stage Key Exchange

In this section, we recall the security model of *multi-stage key exchange (MSKE) protocols*. The model was introduced by [36] and extended in subsequent work [31,33,37,40]. In this paper, we adapt the version presented in [40] almost verbatim apart from the changes discussed in the paragraph below.

Following [40], we describe the MSKE model by specifying *protocol-specific* (Sect. 3.1) and *session-specific* (Sect. 3.2) properties of MSKE protocols as well as the *adversary model* (Sect. 3.3). However, before we start giving the actual model, let us discuss the choice in favor of this model followed by our adaptations to the model.

On the choice of MSKE The most commonly used game-based model for authenticated key exchange goes back to Bellare and Rogaway (BR) [12]. In the context of TLS, it has served as the foundation for the Authenticated and Confidential Channel Establishment (ACCE) model introduced by Jager et al. [43] used for the analyses of TLS 1.2 [43,55], and also the MSKE model initially introduced for analyzing QUIC [36] and later adapted for analyses for TLS 1.3 [31,33,37]. The ACCE model was tailored specifically for the application to TLS 1.2 as it does not allow for a modular analysis due to interleaving of the handshake protocol and record layer. This is because of the established record layer key being already used in the handshake protocol. In TLS 1.3, this was solved by using a dedicated handshake traffic key for the encryption of handshake messages (see Fig. 1) and thus a monolithic model as ACCE is no longer necessary. However, this change introduces another issue. Namely, we now have not only a single key that the communicating parties agree on after the execution of the AKE protocol, but multiple keys being used outside or inside of the protocol. Protocols structured like this motivated Fischlin and Günther (FG) to upgrade the BR model to the MSKE model. Besides the MSKE model, Chen et al. [24] recently proposed a similar ACCE-style model taking into account multiple stages.

We prefer the FG model for an analysis of TLS 1.3 as it is the state-of-the-art security model for TLS 1.3 that is well studied and is already widely used. Most importantly, the model played a major role in the analysis of the Handshake candidates in the standardization process of TLS 1.3. Therefore, using the model in this paper provides the best comparability to previous results on the TLS 1.3 Handshake Protocol. Furthermore, it allows for a modular analysis, i.e., considering the security of the Handshake Protocol and Record Layer in separation. Fischlin and Günther also provide a composition theorem for MSKE protocols (see Sect. 7) allowing for a more general combination with other protocols compared to an ACCE-style model, which only captures secure combination with a encryption protocol.

Indeed, this theorem is very powerful as it allows to argue secure composition with various symmetric key protocol instances. For instance, in the case of the TLS 1.3 Full Handshake the parties exchange an application traffic key to be used in the TLS 1.3 Record Layer, a resumption master secret to be used for deriving a pre-shared key for later session resumption and an exporter master secret to be used as generic keying material exporters [64]. Therefore, the composition theorem allows us to guarantee secure use of all of these keys in their respective symmetric protocols (provided the protocols are secure on their own with respect to some well-defined security notion). In particular, this means that we even have security for a cascading execution of a TLS 1.3 Full Handshake followed by abbreviated PSK Handshakes. For details on the protocol and the composition theorem, see Sects. 4 and 7, respectively.

Changes to the model compared to [40] We only consider the public-key variant of this model, i.e., we exclude pre-shared keys entirely in our model. Since this paper considers TLS 1.3, which does not use semi-static keys in its final version, we also remove these from the original model for simplicity. Further, in the full (EC)DHE TLS 1.3 handshake (Sect. 4) considered in this paper, every stage is *non-replayable*. To that end, we remove the property REPLAY from the *protocol-specific properties* defined in Sect. 3.1. Moreover, TLS 1.3 provides key independence. Therefore, we also remove key-dependent security from the model. Finally, we fix the key distribution $\mathcal{D}$ to be the uniform distribution on $\{0,1\}^\nu$ for some key length $\nu \in \mathbb{N}$.

3.1. *Protocol-Specific Properties*

The protocol-specific properties of a MSKE protocol are described by a vector (M, AUTH, USE) described next. In this section, we consider the properties of the model in general and discuss their concrete instantiation for TLS 1.3 in Sect. 4.3.

- $\mathsf{M} \in \mathbb{N}$ is the *number of stages* the protocol is divided in. This also defines the number of keys derived during the protocol run.
- $\mathsf{AUTH} \subseteq \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}^{\mathsf{M}}$ is the *set of supported authentication types* of the MSKE protocol. An element $\mathsf{auth} \in \mathsf{AUTH}$ describes the mode of authentication for each stage of the key exchange protocol. A stage (resp. the key derived in that stage) is *unauthenticated* if it provides no authentication of either communication partner, *unilaterally authenticated* if it only requires authentication by the responder (server), and *mutually authenticated* if both communication partners are authenticated during the stage.
- $\mathsf{USE} \in \{\mathsf{internal}, \mathsf{external}\}^{\mathsf{M}}$ is the vector describing how derived keys are used in the protocol such that an element USE_i indicates how the key derived in stage i is used. An *internal* key is used within the key exchange protocol and might also be used outside of it. In contrast, an *external* key *must not* be used within the protocol, which makes them amenable to the usage in a protocol used in combination with the key exchange protocol (e.g., symmetric key encryption; see also Sect. 7).

3.2. *Session-Specific Properties*

We consider a set of *users* $\mathcal{U}$ representing the participants in the system and each user is identified by some $U \in \mathcal{U}$. Each user maintains a number of (local) *sessions* of the protocol, which are identified (in the model) by a *unique label* $\mathsf{lbl} \in \mathcal{U} \times \mathcal{U} \times \mathbb{N}$, where $\mathsf{lbl} = (U, V, k)$ indicates the k-th session of user U (*session owner*) with *intended communication partner* V. Each user $U \in \mathcal{U}$ has a *long-term key pair* (pk_U, sk_U), where pk_U is certified.

Also, we maintain a state for each session. Each state is an entry of the *session list* SList and contains the following information:

- $\mathsf{lbl} \in \mathcal{U} \times \mathcal{U} \times \mathbb{N}$ is the *unique session label*, which is only used for administrative reasons in the model.
- $\mathsf{id} \in \mathcal{U}$ is the *identity of the session owner*.
- $\mathsf{pid} \in (\mathcal{U} \cup \{*\})$ is the *identity of the intended communication partner*, where the value $\mathsf{pid} = *$ (wildcard) stands for "unknown identity" and can be set to an identity once during the protocol.
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$ is the *session owner's role* in this session.
- $\mathsf{auth} \in \mathsf{AUTH}$ is the *intended authentication type* for the stages, which is an element of the protocol-specific supported authentication types AUTH.
- $\mathsf{st}_{\mathsf{exec}} \in (\mathsf{RUN} \cup \mathsf{ACC} \cup \mathsf{REJ})$ is the *state of execution*, where

$$\mathsf{RUN} := \{\mathsf{running}_i : i \in \mathbb{N}_0\},\ \mathsf{ACC} := \{\mathsf{accepted}_i : i \in \mathbb{N}_0\},$$
$$\text{and } \mathsf{REJ} := \{\mathsf{rejected}_i : i \in \mathbb{N}_0\}.$$

  With the aid of this variable, the experiment keeps track whether a session can be tested. Namely, a session can only be tested when it just accepted a key and has not used it in the following stage (see Sect. 3.3, Test). Therefore, we set it to one of the following three states: It is set to $\mathsf{accepted}_i$ as soon as a session accepts the i-th key (i.e., it can be tested), to $\mathsf{rejected}_i$ after rejecting the i-th key,[5] and to $\mathsf{running}_i$ when a session continues after accepting key i. The default value is $\mathsf{running}_0$.
- $\mathsf{stage} \in \{0\} \cup [\mathsf{M}]$ is the *current stage*. The default value is 0, and incremented to i whenever $\mathsf{st}_{\mathsf{exec}}$ is set to $\mathsf{accepted}_i$ (resp. $\mathsf{rejected}_i$).
- $\mathsf{sid} \in (\{0,1\}^* \cup \{\perp\})^{\mathsf{M}}$ is the *list of session identifiers*. An element sid_i represents the *session identifier in stage* i. The default value is $\perp$ and it is set once upon acceptance in stage i.
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\perp\})^{\mathsf{M}}$ is the *list of contributive identifiers*. An element cid_i represents the *contributive identifier in stage* i. The default value is $\perp$, and it may be set multiple times until acceptance in stage i.
- $\mathsf{key} \in (\{0,1\}^* \cup \{\perp\})^{\mathsf{M}}$ is the *list of established keys*. An element key_i represents the *established key in stage* i. The default value is $\perp$ and it is set once upon acceptance in stage i.
- $\mathsf{st}_{\mathsf{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}^{\mathsf{M}}$ is *state of the established keys*. An element $\mathsf{st}_{\mathsf{key},i}$ indicates whether the *session key of stage* i *has been revealed* to the adversary. The default value is fresh.

[5] Assumption: The protocol execution halts whenever a stage rejects a key.

– $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^{\mathsf{M}}$ is the *indicator for tested keys*. An element tested_i indicates whether key_i *was already tested* by the adversary. The default value is false.

Shorthands We use shorthands, like $\mathsf{lbl.sid}$, to denote, e.g., the list of session identifiers sid of the entry of SList, which is uniquely defined by label lbl. Further, we write $\mathsf{lbl} \in \mathsf{SList}$ if there is a (unique) tuple $(\mathsf{lbl}, \dots) \in \mathsf{SList}$.

Partnering Following [40], we say that two distinct sessions lbl and lbl' are *partnered* if both sessions hold the same session identifier, i.e., $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid} \neq \perp$. For correctness, we require that two sessions having a non-tampered joint execution are partnered upon acceptance. This means, we consider a MSKE protocol to be correct if, in the absence of an adversary (resp. an adversary that faithfully forwards every message), two sessions running a protocol instance hold the same session identifiers, i.e., they are partnered, upon acceptance.

3.3. *Adversary Model*

We consider an adversary $\mathcal{A}$ that has control over the whole communication network. In particular, that is able to intercept, inject, and drop messages sent between sessions. To model these functionalities, we allow the adversary (as in [40]) to interact with the protocol via the following oracles:

– $\mathsf{NewSession}(U, V, \mathsf{role}, \mathsf{auth})$: Create a new session with a unique new label lbl for session owner $\mathsf{id} = U$ with role role, intended partner $\mathsf{pid} = V$ (might be $V = *$ for "partner unknown"), preferring authentication type $\mathsf{auth} \in \mathsf{AUTH}$. Add $(\mathsf{lbl}, U, V, \mathsf{role}, \mathsf{auth})$ (remaining state information set to default values) to SList and return lbl.

– $\mathsf{Send}(\mathsf{lbl}, m)$: Send message m to the session with label lbl. If $\mathsf{lbl} \notin \mathsf{SList}$, return $\perp$. Otherwise, run the protocol on behalf of $\mathsf{lbl.id}$ on message m, and return both the response and the updated state of execution $\mathsf{lbl.st_{exec}}$. If $\mathsf{lbl.role} = \mathsf{initiator}$ and $m = \top$, where $\top$ denotes the special *initiation symbol*, the protocol initiated and lbl outputs the first message in response.
Whenever the state of execution changes to $\mathsf{accepted}_i$ for some stage i in response to a Send-query, the protocol execution is immediately suspended. This enables the adversary to test the computed key of that stage before it is used in the computation of the response. Using the special $\mathsf{Send}(\mathsf{lbl}, \mathsf{continue})$-query the adversary can resume a suspended session.
If in response to such a query the state of execution changes to $\mathsf{lbl.st_{exec}} = \mathsf{accepted}_i$ for some stage i and there is an entry for a partnered session $\mathsf{lbl}' \in \mathsf{SList}$ with $\mathsf{lbl}' \neq \mathsf{lbl}$ such that $\mathsf{lbl}'.\mathsf{st}_{\mathsf{key},i} = \mathsf{revealed}$, then we set $\mathsf{lbl.st}_{\mathsf{key},i} := \mathsf{revealed}$ as well.[6]

[6] The original model [40] would also handle key dependent security at this point.

If in response to such a query the state of execution changes to $\mathsf{lbl.st_{exec}} = \mathsf{accepted}_i$ for some stage i and there is an entry for a partnered session $\mathsf{lbl}' \in \mathsf{SList}$ with $\mathsf{lbl}' \neq \mathsf{lbl}$ such that $\mathsf{lbl}'.\mathsf{tested}_i = \mathsf{true}$, then set $\mathsf{lbl.tested}_i := \mathsf{true}$ and only if $\mathsf{USE}_i = \mathsf{internal}$, $\mathsf{lbl.key}_i := \mathsf{lbl}'.\mathsf{key}_i$.
If in response to such a query the state of execution changes to $\mathsf{lbl.st_{exec}} = \mathsf{accepted}_i$ for some stage i and $\mathsf{lbl.pid} \neq *$ is corrupted (see Corrupt) by the adversary when lbl accepts, then set $\mathsf{lbl.st}_{\mathsf{key},i} := \mathsf{revealed}$.

– Reveal(lbl, i): Reveal the contents of $\mathsf{lbl.key}_i$, i.e., the session key established by session lbl in stage i, to the adversary.
If $\mathsf{lbl} \notin \mathsf{SList}$ or $\mathsf{lbl.stage} < i$, then return $\perp$. Otherwise, set $\mathsf{lbl.st}_{\mathsf{key},i} := \mathsf{revealed}$ and return the content of $\mathsf{lbl.key}_i$ to the adversary.
If there is a partnered session $\mathsf{lbl}' \in \mathsf{SList}$ with $\mathsf{lbl}' \neq \mathsf{lbl}$ and $\mathsf{lbl}'.\mathsf{stage} \geq i$, then set $\mathsf{lbl}'.\mathsf{st}_{\mathsf{key},i} := \mathsf{revealed}$. Thus, all stage-$i$ session keys of all partnered sessions (if established) are considered to be revealed, too.

– Corrupt(U): Return the long-term secret key sk_U to the adversary. This implies that no further queries are allowed to sessions owned by U after this query. We say that U is *corrupted*.
For stage-j forward secrecy, we set $\mathsf{st}_{\mathsf{key},i} := \mathsf{revealed}$ for each session lbl with $\mathsf{lbl.id} = U$ or $\mathsf{lbl.pid} = U$ and for all $i < j$ or $i > \mathsf{lbl.stage}$. Intuitively, after corruption of user U, we cannot be sure anymore that keys of any stage before stage j as well as keys established in future stages have not been disclosed to the adversary. Therefore, these are considered revealed and we cannot guarantee security for these anymore.

– Test(lbl, i): Test the session key of stage i of the session with label lbl. This oracle is used in the security experiment $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$ given in Definition 10 below and uses a uniformly random test bit b_{Test} as state fixed in the beginning of the experiment definition of $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$.
In case $\mathsf{lbl} \notin \mathsf{SList}$ or $\mathsf{lbl.st_{exec}} \neq \mathsf{accepted}_i$ or $\mathsf{lbl.tested}_i = \mathsf{true}$, return $\perp$. To make sure that key_i has not been used until this query occurs, we set $\mathsf{lost} := \mathsf{true}$ if there is a partnered session lbl' of lbl in SList such that $\mathsf{lbl}'.\mathsf{st_{exec}} \neq \mathsf{accepted}_i$. This also implies that a key can only be tested once (after reaching an accepting state and before resumption of the execution).
We shall only allow the adversary to test a responder session in absence of mutual authentication if this session has an honest (i.e., controlled by the experiment) contributive partner. Otherwise, we would allow the adversary to trivially win the test challenge. Formally, if $\mathsf{lbl.auth}_i = \mathsf{unauth}$, or $\mathsf{lbl.auth}_i = \mathsf{unilateral}$ and $\mathsf{lbl.role} = \mathsf{responder}$, but there is no session $\mathsf{lbl}' \in \mathsf{SList}$ with $\mathsf{lbl}' \neq \mathsf{lbl}$ and $\mathsf{lbl.cid} = \mathsf{lbl}'.\mathsf{cid}$, then set $\mathsf{lost} := \mathsf{true}$.
If the adversary made a valid Test-query, set $\mathsf{lbl.tested}_i := \mathsf{true}$. In case $b_{\mathsf{Test}} = 0$, sample a key $K \xleftarrow{\$} \{0,1\}^{\nu}$ uniformly at random from the session key distribution.[7] In case $b_{\mathsf{Test}} = 1$, set $K := \mathsf{lbl.key}_i$ to be the real session key. If the tested key is an internal key, i.e., $\mathsf{USE}_i = \mathsf{internal}$, set $\mathsf{lbl.key}_i := K$. This means, if the adversary

[7] Note that we replaced the session key distribution $\mathcal{D}$ used in [37,40] by the uniform distribution on $\{0,1\}^{\nu}$, where ν denotes the key length.

gets a random key in response, we substitute the established key by this random key for consistency within the protocol.
Finally, we need to handle partnered session. If there is a partnered session lbl' in SList such that $\mathsf{lbl}.\mathsf{st}_{\mathsf{exec}} = \mathsf{lbl}'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$, i.e., which also just accepted the i-th key, we also set $\mathsf{lbl}'.\mathsf{tested}_i := \mathsf{true}$. We also need to update the state of lbl' in case the established key in stage i is internal. Formally, if $\mathsf{USE}_i = \mathsf{internal}$ then set $\mathsf{lbl}'.\mathsf{key}_i := \mathsf{lbl}.\mathsf{key}_i$. Therefore, we ensured consistent behavior in the further execution of the protocol.
Return K to the adversary.

3.4. *Security Definition*

The security definition of multi-stage key exchange as proposed in [37,40] is twofold. On the one hand, we consider an experiment for *session matching* already used by [23]. In essence, this captures that the specified session identifiers (sid in the model) match in partnered sessions. This is necessary to ensure soundness of the protocol. On the other hand, we consider an experiment to capture classical key indistinguishability transferred into the multi-stage setting. This includes the goals of key independence, stage-j forward secrecy and different modes of authentication.

3.4.1. *Session Matching*

The notion of Match-security according to [40] captures the following properties:

1. Same session identifier for some stage $\implies$ Same key at that stage.
2. Same session identifier for some stage $\implies$ Agreement on that stage's authentication level.
3. Same session identifier for some stage $\implies$ Same contributive identifier at that stage.
4. Sessions are partnered with the indented (authenticated) participant.
5. Session identifiers do not match across different stages.
6. At most two session have the same session identifier at any (non-replayable) stage.

Definition 9. (Match-*Security*) Let KE be a multi-stage key exchange protocol with properties $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE})$, and let $\mathcal{A}$ be an adversary interacting with KE via the oracles defined in Sect. 3.3. Consider the following experiment $\mathsf{Exp}^{\mathsf{Match}}_{\mathsf{KE}}(\mathcal{A})$:

1. The challenger generates a long-term key pair (pk_U, sk_U) for each user $U \in \mathcal{U}$ and hands the public keys $(pk_U)_{U \in \mathcal{U}}$ to the adversary.
2. The adversary may issue queries to the oracles $\mathsf{NewSession}$, Send, Reveal, $\mathsf{Corrupt}$ and Test as defined in Sect. 3.3.
3. Finally, the adversary halts with no output.
4. The experiment outputs 1 if and only if at least one of the following conditions holds:

   (a) *Partnered sessions have different session keys in some stage:* There are two sessions $\mathsf{lbl} \neq \mathsf{lbl}'$ such that for some $i \in [\mathsf{M}]$ it holds $\mathsf{lbl}.\mathsf{sid}_i = \mathsf{lbl}'.\mathsf{sid}_i \neq \perp$, $\mathsf{lbl}.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$ and $\mathsf{lbl}'.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$ but $\mathsf{lbl}.\mathsf{key}_i \neq \mathsf{lbl}'.\mathsf{key}_i$.

(b) *Partnered sessions have different authentication types in some stage:* There are two sessions $\mathsf{lbl} \neq \mathsf{lbl}'$ such that for some $i \in [\mathsf{M}]$ it holds $\mathsf{lbl.sid}_i = \mathsf{lbl}'.\mathsf{sid}_i \neq \perp$, but $\mathsf{lbl.auth}_i \neq \mathsf{lbl}'.\mathsf{auth}_i$.

(c) *Partnered sessions have different or unset contributive identifiers in some stage:* There are two sessions $\mathsf{lbl} \neq \mathsf{lbl}'$ such that for some $i \in [\mathsf{M}]$ it holds $\mathsf{lbl.sid}_i = \mathsf{lbl}'.\mathsf{sid}_i \neq \perp$, but $\mathsf{lbl.cid}_i \neq \mathsf{lbl}'.\mathsf{cid}_i$ or $\mathsf{lbl.cid}_i = \mathsf{lbl}'.\mathsf{cid}_i = \perp$.

(d) *Partnered sessions have a different intended authenticated partner:* There are two sessions $\mathsf{lbl} \neq \mathsf{lbl}'$ such that for some $i \in [\mathsf{M}]$ it holds $\mathsf{lbl.sid}_i = \mathsf{lbl}'.\mathsf{sid}_i \neq \perp$, $\mathsf{lbl.auth}_i = \mathsf{lbl}'.\mathsf{auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$, $\mathsf{lbl.role} = \mathsf{initiator}$, $\mathsf{lbl}'.\mathsf{role} = \mathsf{responder}$, but $\mathsf{lbl.pid} \neq \mathsf{lbl}'.\mathsf{id}$ or in case $\mathsf{lbl.auth}_i = \mathsf{mutual}$, $\mathsf{lbl.id} \neq \mathsf{lbl}'.\mathsf{pid}$.

(e) *Different stages have the same session identifier:* There are two sessions $\mathsf{lbl}, \mathsf{lbl}'$ such that for some $i, j \in [\mathsf{M}]$ with $i \neq j$ it holds $\mathsf{lbl.sid}_i = \mathsf{lbl}'.\mathsf{sid}_j \neq \perp$.

(f) *More than two sessions have the same session identifier in a stage:* There are three pairwise distinct sessions $\mathsf{lbl}, \mathsf{lbl}', \mathsf{lbl}''$ such that for some $i \in [\mathsf{M}]$ it holds $\mathsf{lbl.sid}_i = \mathsf{lbl}'.\mathsf{sid}_i = \mathsf{lbl}''.\mathsf{sid}_i \neq \perp$.

We denote the *advantage of adversary $\mathcal{A}$ in breaking the* Match*-security of* KE by

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE}}(\mathcal{A}) := \Pr[\mathsf{Exp}^{\mathsf{Match}}_{\mathsf{KE}}(\mathcal{A}) = 1]$$

where $\mathsf{Exp}^{\mathsf{Match}}_{\mathsf{KE}}(\mathcal{A})$ denotes the experiment described above.

3.4.2. *Multi-Stage Key Secrecy*

Now, to capture the actual key secrecy, we describe the multi-stage key exchange security experiment. Again, this is adapted from [40].

Definition 10. (MSKE-*Security*) Let KE be a multi-stage key exchange protocol with key length ν and properties (M, AUTH, USE), and let $\mathcal{A}$ be an adversary interacting with KE via the oracles defined in Sect. 3.3. Consider the following experiment $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$:

1. The challenger generates a long-term key pair for each user $U \in \mathcal{U}$ and hands the generated public keys to the adversary. Further, it samples a test bit $b_{\mathsf{Test}} \xleftarrow{\$} \{0, 1\}$ uniformly at random and sets $\mathsf{lost} := \mathsf{false}$.
2. The adversary may issue queries to the oracles NewSession, Send, Reveal, Corrupt and Test as defined in Sect. 3.3. Note that these queries may set the lost flag.
3. Finally, the adversary halts and outputs a bit $b \in \{0, 1\}$.
4. Before checking the winning condition, the experiment checks whether there exist two (not necessarily distinct) labels $\mathsf{lbl}, \mathsf{lbl}'$ and some stage $i \in [\mathsf{M}]$ such that $\mathsf{lbl.sid}_i = \mathsf{lbl}'.\mathsf{sid}_i$, $\mathsf{lbl.st}_{\mathsf{key},i} = \mathsf{revealed}$ and $\mathsf{lbl}'.\mathsf{tested}_i = \mathsf{true}$. If this is the case, the experiment sets $\mathsf{lost} := \mathsf{true}$. This condition ensures that the adversary cannot win the experiment trivially.
5. The experiment outputs 1 if and only if $b = b_{\mathsf{Test}}$ and $\mathsf{lost} = \mathsf{false}$. In this case, we say that *the adversary $\mathcal{A}$ wins the* Test*-challenge*.

We denote the *advantage of adversary* $\mathcal{A}$ *in breaking the* MSKE*-security of* KE by

$$\mathsf{Adv}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A}) := \left| \Pr[\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A}) = 1] - \frac{1}{2} \right|$$

where $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$ denotes the experiment described above.

Remark 2. Note that the winning condition is independent of the required security goals. Key independence, stage-j forward secrecy and authentication properties are defined by the oracles described in Sect. 3.3.

4. TLS 1.3 Full (EC)DHE Handshake

In this section, we describe the cryptographic core of the final version of TLS 1.3 standardized as RFC 8446 [65]. In our view, we do not consider any negotiation of cryptographic parameters. Instead, we consider the cipher suite (AEAD and hash algorithm), the DH group and the signature scheme to be fixed once and for all. In the following, we denote the AEAD scheme by AEAD, the hash algorithm by H, the DH group by $\mathbb{G}$ and the signature scheme by SIG. The output length of the hash function H is denoted by $\mu \in \mathbb{N}$ and the prime order of the group $\mathbb{G}$ by p. The functions HKDF.Extract and HKDF.Expand used in the TLS 1.3 handshake are as defined in Sect. 2.8.[8] Further, we do not consider the session resumption or 0-RTT modes of TLS 1.3.

4.1. *Protocol Description*

The full TLS 1.3 (EC)DHE Handshake Protocol is depicted in Fig. 1.

In the following, we describe the messages exchanged during the handshake in detail. We use the terminology used in the specification RFC 8446 [65]. For further detail we also refer to this specification. Subsequently, we discuss our abstraction of the TLS 1.3 key schedule.

`ClientHello` (`CH`): The `ClientHello` message is the first message of the TLS 1.3 Handshake and is used by a client to initiate the protocol with a server. The message itself consists of five fields. For our analysis the only important one is `random`, which is the random nonce chosen by the client, consisting of a 32-byte value r_C. The remaining values are mostly for backwards compatibility, which is irrelevant for our analysis as we only consider the negotiation of TLS 1.3. There also is a value for the supported ciphersuites of the client, which we omit since we consider the ciphersuite to be fixed once and for all.

[8] The context information *ctx*, i.e., the second parameter of HKDF.Expand is also represented differently in the specification. It just adds constant overhead to the labels which does not harm security and including them would make our view even more complicated. For details, we refer the reader to the TLS 1.3 specification [65].

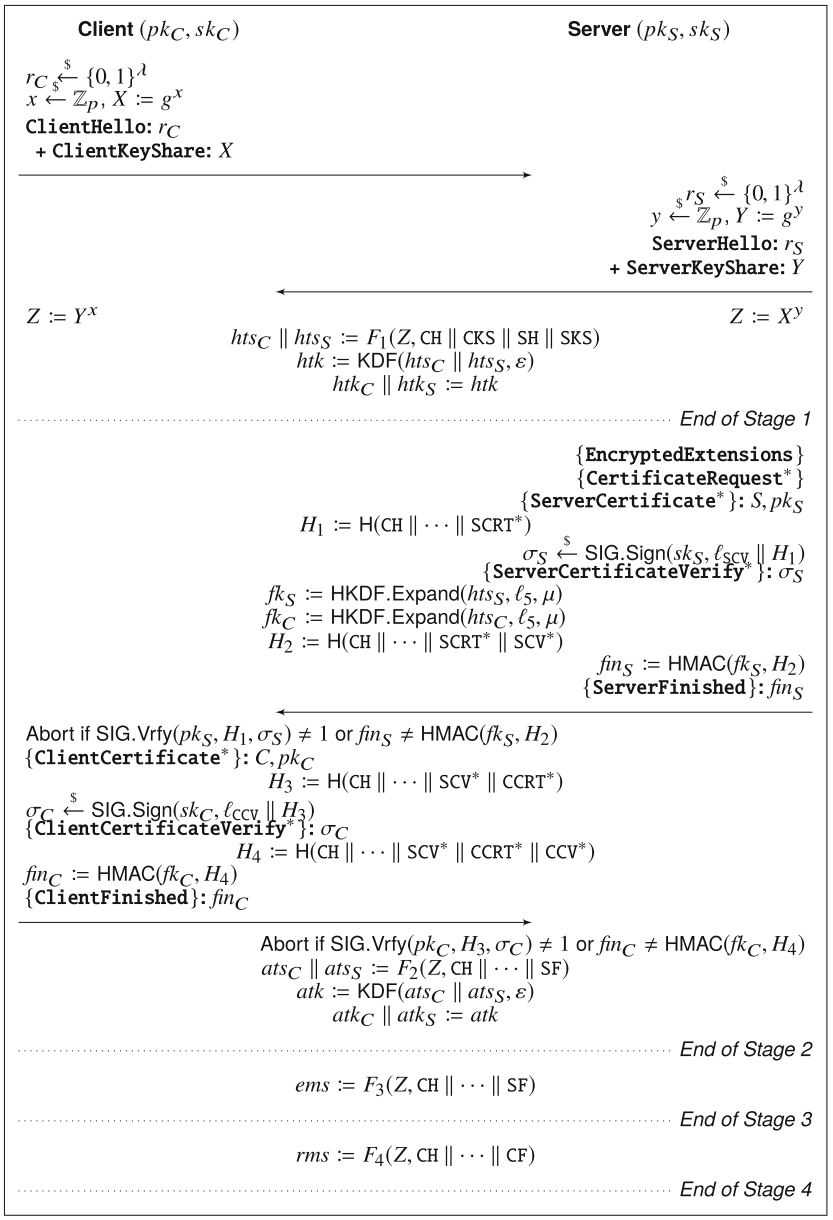

Fig. 1. TLS 1.3 full (EC)DHE handshake. Every TLS handshake message is denoted as "**MSG**: C", where C denotes the message's content. Similarly, an extension is denoted by "**+ MSG**: C". Further, we denote by "{**MSG**}: C" messages containing C and being AEAD-encrypted under the handshake traffic key htk. A message "**MSG***" is an optional, resp. context-dependent message. Centered computations are executed by both client and server with their respective messages received, and possibly at different points in time. The functions KDF, $F_1, \ldots, F_4$ are defined in Figs. 3 and 2, and $\ell_{\mathsf{SCV}} =$ `"TLS 1.3, server CertificateVerify"` and $\ell_{\mathsf{CCV}} =$ `"TLS 1.3, client CertificateVerify"`.

There are various extensions added to this message. For our view only the `key_share` extension is important. We denote this as a separate message called `ClientKeyShare` described next.

`ClientKeyShare` (`CKS`): The `key_share` extension of the `ClientHello` message consists of the public DHE value X chosen by the client. It is defined as $X := g^x$, where $x \xleftarrow{\$} \mathbb{Z}_p$ is the client's private DHE exponent and g the generator of the considered group $\mathbb{G}$. It only contains a single key share as we only consider a single group, which is fixed once and for all before the execution of the protocol.

`ServerHello` (`SH`): In response to the `ClientHello` the server sends the `ServerHello`. This message is structured similarly to the `ClientHello` message. Again, in our view only the `random` field is of importance. Here, we denote the 32-byte random value chosen by the server by r_S.
Similar to the `ClientHello` message there are various extensions added to this message. We only consider the `key_share` extension, which we denote as a separate message `ServerKeyShare` described next.

`ServerKeyShare` (`SKS`) This message consists of the server's public DHE value Y chosen by the server. It is defined as $Y := g^y$, where $y \xleftarrow{\$} \mathbb{Z}_p$ is the server's private DHE exponent and g the generator of $\mathbb{G}$.

After this message is computed, the server is ready to compute the *handshake traffic key htk*. To that end, the server first computes the exchanged DHE key $Z := X^y$, where X is the client's public DHE value sent in the `ClientKeyShare` message. Using Z and the handshake messages computed and received so far, i.e., `CH`, `CKS`, `SH`, `SKS`, it computes the *handshake secret hs*, the *client handshake traffic secret* hts_C and the *server handshake traffic secret* hts_S. In our abstraction this is done by evaluating the function F_1 defined in Fig. 2, where *hs* is only computed internally. Formally,

$$hts_C \parallel hts_S := F_1(Z, \texttt{CH} \parallel \texttt{CKS} \parallel \texttt{SH} \parallel \texttt{SKS}).$$

Based on the handshake traffic secrets hts_C and htk_S the server derives the *handshake traffic key*

$$htk := \mathsf{KDF}(hts_C \parallel hts_S, \varepsilon).$$

The definition of KDF is given in Fig. 3. In essence, it summarizes the traffic key derivation in the way that encryption key and initialization vector (IV) are now abstracted into a single key and also combines the derivation for both parties into a single function call.

The function KDF is not described in the specification [65]. We introduce this function to tame the complexity of our security proof.[9] We discuss the security of KDF in Sect. 5.3.

Upon receiving (`SH`, `SKS`), the client performs the same computations to derive *htk* except that it computes the DHE key as $Z := Y^x$.

[9] Using this function we can reduce the number of games introduced in the security proofs. For details, see Sect. 6.

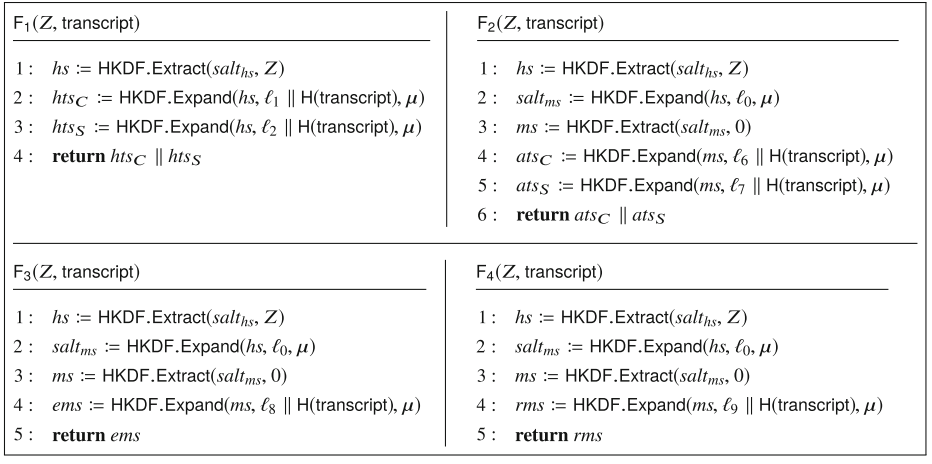

Fig. 2. Definition of the functions F_1, F_2, F_3 and F_4 used in Fig. 1, where $\ell_0 := \texttt{"derived"} \,\|\, \mathsf{H}(\varepsilon)$, $salt_{hs} := \mathsf{HKDF.Expand}(es, \ell_0, \mu)$ with $es := \mathsf{HKDF.Extract}(0, 0)$, $\ell_1 := \texttt{"c hs traffic"}$, $\ell_2 := \texttt{"s hs traffic"}$, $\ell_6 := \texttt{"c ap traffic"}$, $\ell_7 := \texttt{"s ap traffic"}$, $\ell_8 := \texttt{"exp master"}$, and $\ell_9 := \texttt{"res master"}$.

$\mathsf{KDF}(s_1 \parallel s_2, m)$

1 : $k_1 := \mathsf{HKDF.Expand}(s_1, \ell_3 \parallel m, l)$
2 : $iv_1 := \mathsf{HKDF.Expand}(s_1, \ell_4 \parallel m, d)$
3 : $k_2 := \mathsf{HKDF.Expand}(s_2, \ell_3 \parallel m, l)$
4 : $iv_2 := \mathsf{HKDF.Expand}(s_2, \ell_4 \parallel m, d)$
5 : **return** $(iv_1, k_1) \parallel (iv_2, k_2)$

Fig. 3. Definition of the function KDF used in Fig. 1. Let $s_1, s_2 \in \{0,1\}^\mu$, where μ is the output length of the hash function used as a subroutine of HKDF.Expand, let $m \in \{0,1\}^*$ and let $l, d \in \mathbb{N}$ with l being the encryption key length and d being the IV length of AEAD, respectively. Further, let $\ell_3 :=$ `"key"` and let $\ell_4 :=$ `"iv"`.

All following messages sent from now on are encrypted under the handshake traffic key *htk* using AEAD. For the direction "server → client", we use the *server handshake traffic key* htk_S and for the opposite direction, we use the *client handshake traffic key* htk_C.

`EncryptedExtensions` (`EE`): This message contains all extensions that are not required to determine the cryptographic parameters. In previous versions, these extensions were sent in the plain. In TLS 1.3, these extensions are encrypted under the server handshake traffic key htk_S.

`CertificateRequest` (`CR`): The `CertificateRequest` message is a context-dependent message that may be sent by the server. The server sends this message when it desires client authentication via a certificate.

`ServerCertificate` (`SCRT`): This context dependent message consists of the actual certificate of the server used for authentication against the client. Since we do not consider any PKI, we view this message as some certificate[10] that contains some server identity S and a public key pk_S appropriate for the signature scheme.

`ServerCertificateVerify` (`SCV`): To provide a "proof" that the server sending the `ServerCertificate` message really is in possession of the private key sk_S corresponding to the announced public key pk_S, it sends a signature $\sigma_S \xleftarrow{\$} \mathsf{SIG.Sign}(sk_S, \ell_{\mathtt{SCV}} \parallel H_1)$ over the hash H_1 of the messages sent and received so far, i.e.,

$$H_1 = \mathsf{H}(\mathtt{CH} \parallel \mathtt{CKS} \parallel \mathtt{SH} \parallel \mathtt{SKS} \parallel \mathtt{EE} \parallel \mathtt{CR}^* \parallel \mathtt{SCRT}^*).$$

This message is only sent when the `ServerCertificate` message was sent. Recall that every message marked with * is an optional or context-dependent message.

`ServerFinished` (`SF`): This message contains the HMAC (Sect. 2.7) value over a hash of all handshake messages computed and received by the server. To that end,

[10] The certificate might be self-signed.

the server derives the *server finished key* fk_S from hts_S as $fk_S := \mathsf{HKDF.Expand}(hts_S, \ell_5, \mu)$, where $\ell_5 :=$ `"finished"` and $\mu \in \mathbb{N}$ denotes the output length of the used hash function H. Then, it computes the MAC

$$fin_S := \mathsf{HMAC}(fk_S, H_2)$$

with $H_2 = \mathsf{H}(\mathtt{CH} \parallel \mathtt{CKS} \parallel \mathtt{SH} \parallel \mathtt{SKS} \parallel \mathtt{EE} \parallel \mathtt{CR}^* \parallel \mathtt{SCRT}^* \parallel \mathtt{SCV}^*)$.

Upon receiving (and after decryption) of ($\mathtt{EE}$, $\mathtt{CR}^*$, $\mathtt{SCRT}^*$, $\mathtt{SCV}^*$), the client first checks whether the signature and MAC contained in the `ServerCertificateVerify` message and `ServerFinished` message, respectively, are valid. To that end, it retrieves the server's public key from the `ServerCertificate` message (if present), derives the server finished key based on hts_S, and recomputes the hashes H_1 and H_2 with the messages it has computed and received. The client aborts the protocol if either of the message are not sound. Provided the client does not abort, it prepares the following messages.

`ClientCertificate` ($\mathtt{CCRT}$): This message is context-dependent and is only sent by the client in response to a `CertificateRequest` message, i.e., if the server demands client authentication. The message is structured analogously to the `ServerCertificate` message except that it contains a client identity C and an appropriate public key pk_C.

`ClientCertificateVerify` ($\mathtt{CCV}$): This message also is context-dependent and only sent in conjunction with the `ClientCertificate` message. Similar to message `ServerCertificateVerify`, this message contains a signature $\sigma_C \xleftarrow{\$} \mathsf{SIG.Sign}(sk_C, \ell_{\mathrm{CCV}} \parallel H_3)$ over the hash H_3 of all messages computed and received by the client so far, i.e.,

$$H_3 = \mathsf{H}(\mathtt{CH} \parallel \mathtt{CKS} \parallel \mathtt{SH} \parallel \mathtt{SKS} \parallel \mathtt{EE} \parallel \mathtt{CR}^* \parallel \mathtt{SCRT}^* \parallel \mathtt{SCV}^* \parallel \mathtt{CCRT}^*).$$

`ClientFinished` ($\mathtt{CF}$): The last handshake message is the finished message of the client. As for the `ServerFinished` message this message contains a MAC over every message computed and received so far by the client. The client derives the *client finished key* fk_C from hts_C as $fk_C := \mathsf{HKDF.Expand}(hts_C, \ell_5, \mu)$ and then, computes

$$fin_C := \mathsf{HMAC}(fk_C, H_4)$$

with $H_4 = \mathsf{H}(\mathtt{CH} \parallel \mathtt{CKS} \parallel \mathtt{SH} \parallel \mathtt{SKS} \parallel \mathtt{EE} \parallel \mathtt{CR}^* \parallel \mathtt{SCRT}^* \parallel \mathtt{SCV}^* \parallel \mathtt{CCRT}^* \parallel \mathtt{CCV}^*)$.

Upon receiving (and after decryption) of ($\mathtt{CCRT}^*$, $\mathtt{CCV}^*$, $\mathtt{CF}$), the server first checks whether the signature and MAC contained in the `ClientCertificateVerify` message and `ClientFinished` message, respectively, are valid. To that end, it retrieves the client's public key from the `ClientCertificate` message (if present), derives the client finished key based on hts_C, and recomputes the hashes H_3 and H_4 with the messages it received. If one of the checks fails, the server aborts. Otherwise, client

and server are ready to derive the *application traffic key atk*, which is used in the TLS Record Protocol.

They first derive the *master secret ms* from the handshake secret *hs* derived earlier. Based on *ms* and the handshake transcript up to the `ServerFinished` message, client and server derive the *client application traffic secret* ats_C and *server application traffic secret* ats_S, respectively. In our abstraction, ats_C and ats_S are computed by evaluating the function F_2 defined in Fig. 2, i.e.,

$$ats_C \parallel ats_S := F_2(Z, \texttt{CH} \parallel \cdots \parallel \texttt{SF})$$

where *ms* again is computed internally. Using ats_C and ats_S, they finally can derive the *application traffic key*

$$atk := \mathsf{KDF}(ats_C \parallel ats_S, \varepsilon),$$

where KDF (Fig. 3) is the same function used in the derivation of *htk*.

After having derived *atk*, they derive the *exporter master secret ems* from the master secret derived earlier and the handshake transcript up to the `ServerFinished` message. In our abstraction, they evaluate the function F_3 defined in Fig. 2, i.e.,

$$ems := F_3(Z, \texttt{CH} \parallel \cdots \parallel \texttt{SF}).$$

Finally, they derive *resumption master secret rms* from the master secret derived earlier and the handshake transcript up to the `ClientFinished` message. In our abstraction, they evaluate the function F_4 defined in Fig. 2, i.e.,

$$rms := F_4(Z, \texttt{CH} \parallel \cdots \parallel \texttt{CF}).$$

4.2. *On Our Abstraction of the TLS 1.3 Key Schedule*

In our presentation of the TLS 1.3 Handshake Protocol, we decompose the TLS 1.3 Key Schedule [65, Sect. 7.1] into independent key derivation steps. The main reason for this abstraction is a technical detail of the proof presented in Sect. 6, but also the nature of the MSKE security model requires a key derivation in stages to enable testing the stage keys before possible internal usage of them. Therefore, we introduce a dedicated function for every stage key derivation. These functions are F_1, F_2, F_3 and F_4 defined in Fig. 2. Considering the definition of these functions, they seem quite redundant as values, like the handshake secret *hs*, are computed multiple times. We stress that this is only conceptual and does not change the implementation of the original TLS 1.3 Handshake Protocol. When running the protocol, these values of course can be cached and reused in following computations. We need this modularized key derivation as we will model each of these derivation steps as a random oracle in our proof. For completeness, we give the TLS 1.3 Key Schedule as it is defined in the standard in Fig. 4. Our decomposition essentially consists of viewing the derivations of hts_C/hts_S, ats_C/ats_S, *ems* and *rms* as separate functions based on the DHE key Z and the transcript.

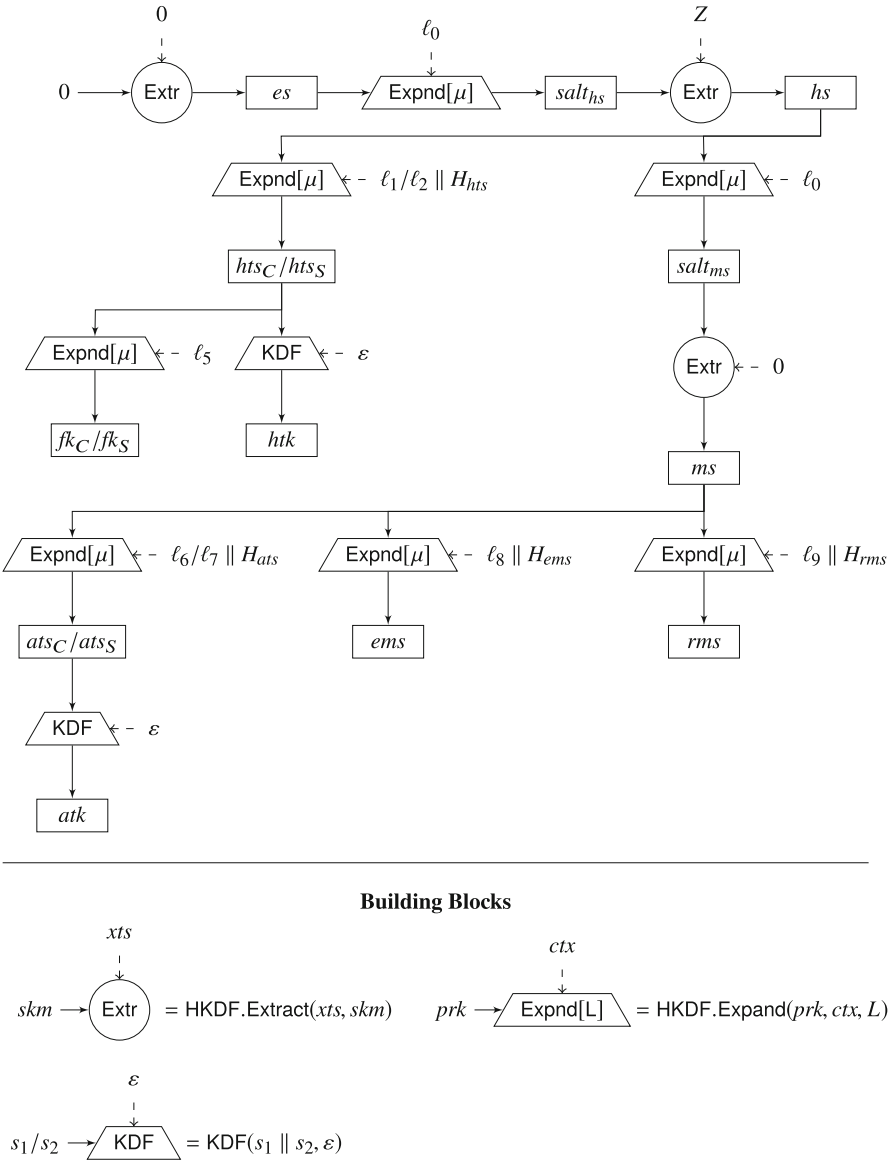

Fig. 4. TLS 1.3 Key Schedule. The labels ℓ_i are defined in Sect. 4.1. The hash values are defined as $H_{hts} = \mathsf{H}(\mathtt{CH} \parallel \cdots \parallel \mathtt{SKS})$, $H_{ats} = H_{ems} = \mathsf{H}(\mathtt{CH} \parallel \cdots \parallel \mathtt{SF})$, and $H_{rms} = \mathsf{H}(\mathtt{CH} \parallel \cdots \parallel \mathtt{CF})$. In this picture, we use the notation v_1/v_2 to denote alternative usage of v_1 and v_2 in analogous computations .

4.3. *TLS 1.3 Handshake Protocol as a Multi-stage Key Exchange Protocol*

In this section, we model the TLS 1.3 Handshake as presented before as a multi-stage key exchange protocol. In particular, this means to define the protocol-specific properties as given in Sect. 3.1 and to describe how, e.g., session and contributive identifiers, are defined. We follow [40] and adapt it to the current version of TLS 1.3 given in [65].

Protocol-Specific Properties The TLS 1.3 Handshake has the following protocol-specific properties ($\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}$):

1. $\mathsf{M} = 4$: In the full TLS 1.3 Handshake there are four keys derived, which are the handshake traffic key *htk* in stage 1, the application traffic key *atk* in stage 2, the resumption master secret *rms* in stage 3, and the exporter master secret *ems* in stage 4.
2. $\mathsf{AUTH} = \{(\mathsf{unauth}, \mathsf{auth}', \mathsf{auth}', \mathsf{auth}') : \mathsf{auth}' \in \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}\}$: The handshake traffic key derived in stage 1 is always unauthenticated. The keys derived in stages 2–4 can *all* either be unauthenticated, server-only, or mutually authenticated. Note that our result given in Theorem 6 covers all of these authentication types.
3. $\mathsf{USE} = (\mathsf{internal}, \mathsf{external}, \mathsf{external}, \mathsf{external})$: The handshake traffic key is used internally during the handshake, whereas all the other keys derived are only used outside the full handshake.

We define the session identifiers for each stage (analogously to [40]) as follows:

- $\mathsf{sid}_1 = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS})$
- $\mathsf{sid}_2 = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{CR}^*, \mathtt{SCRT}^*, \mathtt{SCV}^*, \mathtt{CCRT}^*, \mathtt{CCV}^*)$
- $\mathsf{sid}_3 = (\mathsf{sid}_2, \text{“RMS”})$
- $\mathsf{sid}_4 = (\mathsf{sid}_3, \text{“EMS”})$

Note that each message marked with $*$ is context dependent and might not be present, e.g., depending on the used authentication types.

Further, we set $\mathsf{cid}_1 := (\mathtt{CH}, \mathtt{CKS})$ after the client sends (resp. the server receives) these messages. After the server sent its `ServerHello` message (resp. the client receives it), we extend $\mathsf{cid}_1 := (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}) = \mathsf{sid}_1$. The contributive identifiers for stage 2–4 are set by each party after sending its respective finished message to $\mathsf{cid}_i := \mathsf{sid}_1$.

4.3.1. Match-*Security of TLS 1.3 Handshake*

The proof of Match-security of the TLS 1.3 Handshake Protocol as described above basically follows along the lines of the proof given by [40, Thm. 6.1] for TLS 1.3 `draft-10`. We restate the proof and adapt it to the final version of TLS 1.3 Handshake.

Theorem 1. *Let $\lambda \in \mathbb{N}$. Let $\mathcal{A}$ be an adversary against the* Match*-security of the TLS 1.3 Handshake protocol as described in Sect. 4. Then, we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathrm{TLS1.3}}(\mathcal{A}) \leq \frac{n_s^2}{p \cdot 2^{\lambda}}$$

where n_s is the maximum number of sessions, p is the order of $\mathbb{G}$ used in the handshake and $\lambda = 256$ is the bit-length of the nonces.

Intuitively, this bound is the probability that there are two sessions that choose the same nonce and the same key share.

Proof. In order to prove this statement, we need to show each of the properties stated in Definition 9.

1. *Same session identifier for some stage $\implies$ same key at that stage*: For stage 1, we have that the session identifier sid_1 uniquely defines the ephemeral DH key Z, as it contains the public DH values contained in CKS and SKS, respectively. Further, Z and all messages contained in sid_1, i.e., (CH, CKS, SH, SKS), uniquely define the values *hs*, hts_C and hts_S. The values hts_C and hts_S, in turn, uniquely define the stage-1 key *htk*. For the remaining stage, first note that sid_2 completely contains sid_1. As sid_2 is completely contained in sid_3 and sid_4, sid_1 is also contained in sid_3 and sid_4. The key derivation in stages 2–4 (i.e., the derivation of *atk*, *ems* and *rms*) solely depends on *ms* and the messages contained in sid_2. As sid_2 contains sid_1, the parties compute the same *hs*, hts_C and hts_S (see above). The handshake secret *hs* uniquely defines the value *ms*. Then, hts_C and hts_S define the server's and client's finished keys. Using these keys and the messages contained in sid_2 the (valid) messages SF and CF (depending on SF) are uniquely defined. Finally, taking all this together the computations of *atk*, *ems* and *rms* are uniquely defined by sid_2.
2. *Same session identifier for some stage $\implies$ agreement on that stage's authentication level*: The first stage is always authenticated. Therefore, $\mathsf{auth}_i = \mathsf{unauth}$ for *all* sessions. For stages 2–4, recall that sid_2 is contained completely in sid_3 and sid_4. Moreover, recall that sid_2 contains each handshake message exchanged apart from the finished messages, which are defined by the messages contained in sid_2. Therefore, sid_2 reflects the used authentication types by the presence of the context-dependent messages. That is, if $\mathsf{sid}_2 = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE})$ then $\mathsf{auth}_2 = \mathsf{unauth}$. If in addition the message $\mathtt{SCRT}^*$, $\mathtt{SCV}^*$ are present in sid_2, we have $\mathsf{auth}_2 = \mathsf{unilateral}$. While we have $\mathsf{auth}_2 = \mathsf{mutual}$ if the messages $\mathtt{CR}^*$, $\mathtt{SCRT}^*$, $\mathtt{SCV}^*$, $\mathtt{CCRT}^*$, $\mathtt{CCV}^*$ are in addition present. Finally, observe that $\mathsf{auth}_2 = \mathsf{auth}_3 = \mathsf{auth}_4$ holds for every session.
3. *Same session identifier for some stage $\implies$ same contributive identifier at that stage*: This is given by definition of the contributive identifier. First, note that sid_1 is contained in the session identifier of every stage. The contributive identifier, for each stage, is set to sid_1 once it is set and never changed.
4. *Sessions are partnered with the indented (authenticated) participant*: This can only be achieved in unilaterally or mutually authenticated stages. This means, this can only be given for stages 2–4. The sessions will obtain the partner identity in a certificate message. That is, in case of unilateral and mutual authentication the client will get the server's identity in the SCRT message. While the server will get the client's identity in the CCRT message in case of mutual authentication. Certificates always belong to a party that are considered honest and honest parties

never send certificates that belong to another identity. As sid_2 (contained in sid_3 and sid_4) contains both certificate message (if present), agreement on sid_2 implies partner agreement as well.

5. *Session identifiers do not match across different stages*: This holds trivially. sid_2 contains strictly less values than sid_1 and both sid_3 and sid_4 contain a dedicated unique label.
6. *At most two session have the same session identifier at any stage*: We analyze the probability that there are three sessions sharing the same identifier. To that end, assume there is a client (initiator) session and a server (responder) session holding the same session identifier for each stage. Due to the certificates included, there only can exist a session of either the client's or the server's party. Note that the sid_1 is contained in every other stages' session identifier and sid_1 defines the other messages contained in the other identifiers (apart from the unique labels and the certificates). The third session therefore needs to sample the same group element and the same nonce as one of the other two sessions. The probability that this happens is bounded from above by

$$\frac{n_s^2}{p \cdot 2^{\lambda}}$$

which is the probability that both group element and nonces collide for any two sessions out of n_s, where n_s is the maximum number of sessions.

5. Tight Security of TLS 1.3 PRFs

In this section, we consider the tight security of the PRFs used in TLS 1.3.

5.1. *Tight Security of* HMAC

[6] has shown that the HMAC function (i.e., MAC.Tag as presented in Sect. 2.7) is a PRF as long as the used function H is a (dual) PRF. In this paper, we show that HMAC is tightly MU−PRF-secure (Definition 4) in the random oracle model.

Theorem 2. *Let $\kappa, \mu \in \mathbb{N}$ and let* H *be modeled as a random oracle with output length μ. Further, let $\mathcal{A}$ be an adversary against the* MU–PRF*-security with N users of* HMAC*. Then,*

$$\mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HMAC},N}(\mathcal{A}) \leq \frac{N^2}{2^{\kappa}} + \frac{q_{\mathsf{H}}^2}{2^{\mu}} + \frac{2N}{2^{\kappa}}$$

where κ is the key length of the HMAC *function and q_{H} is the number of queries issued to the random oracle* H*.*

For simplicity, we prove this statement under the assumption that $\kappa \leq 8B$, where B is the byte-length of opad and ipad, respectively. In the context of TLS 1.3, this

assumption is reasonable as all ciphersuites either use SHA–256 or SHA–384 [65, Appx. B4] as their hash algorithm, where SHA–256 has a block length of $B = 64$ bytes and SHA–384 a block length of $B = 128$ bytes. In TLS 1.3 (Sect. 4), we have $\kappa = \mu$ for every direct application of HMAC, i.e., including when ran as a subroutine of HKDF.Extract or HKDF.Expand. That is, for SHA–256 and SHA–384 we always have $\mu = 256$ and $\mu = 384$ bits, respectively.

However, the proof can simply be extended by the case $\kappa > B$ by adding another step.

Proof. We prove this statement in a sequence of games [68]. Let break_δ denote the event that $\mathcal{A}$ wins in Game δ, i.e., Game δ outputs 1, and let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta] - 1/2$.

Game 0 The initial game is the multi-user PRF experiment $\mathsf{Exp}^{\mathsf{MU-PRF}}_{\mathsf{HMAC},N}(\mathcal{A})$ given in Definition 4 and thus

$$\Pr[\mathsf{break}_0] = \Pr[\mathsf{Exp}^{\mathsf{MU-PRF}}_{\mathsf{HMAC},N}(\mathcal{A}) = 1] = \frac{1}{2} + \mathsf{Adv}_0.$$

Game 1 In this game, we make sure that every user has a distinct key. To that end, we add an abort rule and raise the event $\mathsf{abort}_{\mathsf{key}}$ if there are keys k_i, k_j such that $k_i = k_j$ for users $i, j \in [N]$ with $i \neq j$. Game 0 and Game 1 are identical until the event $\mathsf{abort}_{\mathsf{key}}$ occurs. Thus, we have by the Difference Lemma ([68, Lem. 1]) that

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \Pr[\mathsf{abort}_{\mathsf{key}}].$$

It remains to analyze $\Pr[\mathsf{abort}_{\mathsf{key}}]$. The event $\mathsf{abort}_{\mathsf{key}}$ is raised if there is a collision among N uniform and independent samples from the set $\{0,1\}^\kappa$. An upper bound for this probability is given by the birthday bound, which supplies

$$\Pr[\mathsf{abort}_{\mathsf{key}}] \leq \frac{N^2}{|\{0,1\}^\kappa|} = \frac{N^2}{2^\kappa}.$$

Hence, $\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + N^2/2^\kappa$.

Note that if the game is not aborted due to this rule, we have that the strings $k_i \oplus \mathsf{ipad}$ and $k_i \oplus \mathsf{opad}$ are distinct for every $i \in [N]$. Thus, in case the challenge bit is $b = 0$, i.e., the oracle is the real PRF, there are no two queries (i, x) and (j, x) with $i \neq j$ to oracle $\mathcal{O}_b$ such that the challenger uses the same prefix $k_i \oplus \mathsf{opad} = k_j \oplus \mathsf{opad}$ in the computation of the respective oracle responses

$$\mathsf{H}((k_i \oplus \mathsf{opad}) \parallel \mathsf{H}((k_i \oplus \mathsf{ipad}) \parallel x)) \quad \text{and} \quad \mathsf{H}((k_j \oplus \mathsf{opad}) \parallel \mathsf{H}((k_j \oplus \mathsf{ipad}) \parallel x)).$$

The same applies to the prefix of the internal random oracle call.

Game 2 This game is identical to Game 1, except that we add another abort rule. If during the execution of the security experiment the random oracle is ever queried with $h \neq h'$ such that $\mathsf{H}(h) = \mathsf{H}(h')$, we raise the event $\mathsf{abort}_{\mathsf{coll}}$. Since Game 1 and Game

2 are identical until $\mathsf{abort}_{\mathsf{coll}}$ occurs, we have

$$\mathsf{Adv}_1 \leq \mathsf{Adv}_2 + \Pr[\mathsf{abort}_{\mathsf{coll}}].$$

Due to the birthday bound, the probability that a collision in the random oracle occurs is upper bounded by $q_{\mathsf{H}}^2 \cdot 2^{-\mu}$, where q_{H} is the number of queries to the random oracle and μ is the output length of the random oracle. Therefore,

$$\mathsf{Adv}_1 \leq \mathsf{Adv}_2 + \frac{q_{\mathsf{H}}^2}{2^{\mu}}.$$

This modification ensures that, in case $b = 0$, there are no two queries (i, x) and (i, x') with $x \neq x'$ to oracle $\mathcal{O}_b$ such that the responses are computed using

$$\mathsf{H}((k_i \oplus \mathsf{opad}) \parallel \mathsf{H}((k_i \oplus \mathsf{ipad}) \parallel x)) \quad \text{and} \quad \mathsf{H}((k_i \oplus \mathsf{opad}) \parallel \mathsf{H}((k_i \oplus \mathsf{ipad}) \parallel x'))$$

and it holds $\mathsf{H}((k_i \oplus \mathsf{ipad}) \parallel x) = \mathsf{H}((k_i \oplus \mathsf{ipad}) \parallel x')$, i.e., the inner evaluation of the random oracle collide. Otherwise, the responses would also collide as the random oracle needs to ensure consistency.

Game 3 This game is identical to Game 2, except that we add another abort rule. Namely, we want to ensure that the adversary for any $i \in [N]$ never queries the random oracle for the string $pre \parallel y$ such that $pre = k_i \oplus \mathsf{opad}$ or $pre = k_i \oplus \mathsf{ipad}$, and $y \in \{0, 1\}^*$. To that end, we raise the event $\mathsf{abort}_{\mathsf{guess}}$ if the adversary makes such a query. Since Game 2 and Game 3 are identical until $\mathsf{abort}_{\mathsf{guess}}$ occurs, we have

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \Pr[\mathsf{abort}_{\mathsf{guess}}].$$

To analyze $\mathsf{abort}_{\mathsf{guess}}$, let $\mathsf{abort}_{\mathsf{guess;i}}$ be the same event as $\mathsf{abort}_{\mathsf{guess}}$ except that we only consider user i. Then, we have that $\Pr[\mathsf{abort}_{\mathsf{guess}}] = \sum_{i \in [N]} \Pr[\mathsf{abort}_{\mathsf{guess;i}}]$. It holds

$$\begin{aligned}
\Pr[\mathsf{abort}_{\mathsf{guess;i}}] &= \Pr[pre = k_i \oplus \mathsf{opad} \vee pre = k_i \oplus \mathsf{ipad}] \\
&= \Pr[k_i = pre \oplus \mathsf{opad} \vee k_i = pre \oplus \mathsf{ipad}] \\
&= 2 \cdot \Pr[k_i = pre \oplus \mathsf{opad}] = \frac{2}{2^{\kappa}}.
\end{aligned}$$

Hence,

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \frac{2N}{2^{\kappa}}.$$

The main consequence of this modification is that the adversary is not able to query the random oracle with a bit string that at some point may be queried by the challenger in case $b = 0$ to compute a response to any query $\mathcal{O}_b(i, x)$ as otherwise the game would be aborted. However, Games 2 and 3 also remove the side channel of the adversary to

successfully guess a key for some user $i \in [N]$ and compare the outputs of the random oracle and the oracle $\mathcal{O}_b$ to win the game.

If Game 3 is not aborted due to $\mathsf{abort}_{\mathsf{guess}}$, we claim that $\mathsf{Adv}_3 = 0$. To show this, we argue that the answer of the oracle $\mathcal{O}_b$ adversary $\mathcal{A}$ is provided with in $\mathsf{Exp}^{\mathsf{MU\text{–}PRF}}_{\mathsf{HMAC},N}(\mathcal{A})$ is distributed uniformly at random on $\{0,1\}^\mu$ independent of the challenge bit b. To that end, it suffices to argue that this holds in case $b = 0$. In case $b = 1$, this is true by definition.

Games 1, 2 and 3 make sure that every computation of the oracle $\mathcal{O}_0$ is done by a *fresh* random oracle query, i.e., the query was neither issued by the adversary nor by the challenger at any point in time before this query. Consequently, every response of $\mathcal{O}_0$ is a bit string sampled uniformly and independently at random for any query (i, x). Hence, the answer of the oracle is a uniform and independent bit string in case $b = 0$ and $b = 1$, respectively. Formally, the advantage of the adversary in breaking the MU–PRF-security of HMAC is 0 in this setting. □

5.2. *Tight Security of* HKDF

By definition of HKDF.Extract (Sect. 2.8), we get the following result.

Theorem 3. *Let* HMAC *and* HKDF.Extract *be the functions as defined in Sects. 2.7 and 2.8, respectively. Further, let $\mathcal{A}$ be an adversary against the* MU–PRF*-security with N users of* HKDF.Extract*. Then,*

$$\mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Extract},N}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HMAC},N}(\mathcal{A}).$$

Theorem 3 follows from the definition of HKDF.Extract. For HKDF.Expand, we get a similar result.

Theorem 4. *Let* HMAC *and* HKDF.Expand *(with fixed output length L) be the functions as defined in Sects. 2.7 and 2.8, respectively. Further, let $\mathcal{A}$ be an adversary against the* MU−PRF*-security with N users of* HKDF.Expand *running in time at most t. Then, we can construct an adversary $\mathcal{B}$ running in time at most $t' \approx t$ such that*

$$\mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Expand},N}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HMAC},N}(\mathcal{B}).$$

Proof. (Sketch) The proof of Theorem 4 is straightforward. The adversary $\mathcal{B}$ can perfectly simulate the experiment $\mathsf{Exp}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Expand},N}(\mathcal{A})$ by computing every query issued by $\mathcal{A}$ using its oracle. For every query of $\mathcal{A}$, it computes the HKDF.Expand function except that it used its oracle instead of the HMAC function. To that end, for every query of $\mathcal{A}$, $\mathcal{B}$ needs to make $\lceil L/\mu \rceil$ queries to its oracle. In case $\mathcal{B}$ is provided with the real HMAC function, it is easy to see that it perfectly simulates HKDF.Expand. Otherwise, if it is provided with a random function, each of the query answers is distributed uniformly and independently at random. Therefore, the string $\mathcal{A}$ is provided with in response to a query is also a uniformly random string. Hence, if $\mathcal{A}$ wins its experiment, $\mathcal{B}$ also wins.

Summary. Taking these results together with Theorem 2, we get that both HKDF.Extract and HKDF.Extract are tightly MU−PRF-secure (Definition 4) in the random oracle model.

Corollary 1. *Let* HKDF.Extract *(with fixed output length L) be the function as defined in Sect. 2.8 and let* H *(i.e., the internal hash function) be modeled as a random oracle with output length μ. Then, for any adversary $\mathcal{A}$*

$$\mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Extract},N}(\mathcal{A}) \leq \frac{N^2}{2^\kappa} + \frac{q_\mathsf{H}^2}{2^\mu} + \frac{2N}{2^\kappa}$$

where κ is the key length of the HKDF.Extract *function and q_H is the number of queries issued to the random oracle* H.

Corollary 2. *Let* HKDF.Expand *(with fixed output length L) be the function as defined in Sect. 2.8 and let* H *(i.e., the internal hash function) be modeled as a random oracle with output length μ. Then, for any adversary $\mathcal{A}$*

$$\mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Expand},N}(\mathcal{A}) \leq \frac{N^2}{2^\kappa} + \frac{q_\mathsf{H}^2}{2^\mu} + \frac{2N}{2^\kappa}$$

where κ is the key length of the HKDF.Expand *function and q_H is the number of queries issued to the random oracle* H.

5.3. *Security of* KDF

In Sect. 4, we introduced the function KDF (Fig. 3), which combines several computation steps of the TLS 1.3 Handshake Protocol into a single function call. It remains to argue about its security guarantees. KDF uses HKDF.Expand (Sect. 2.8) as a subroutine. In the following, we give a bound for KDF in Theorem 5.

Theorem 5. *Let* HKDF.Expand *be as defined in Sect. 2.8 and* KDF *be as defined in Fig. 3. Then, for any adversary $\mathcal{A}$ against the* MU−PRF*-security of* KDF *running in time at most t, we can construct adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_4$ all running in time at most $t' \approx t$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{KDF},N}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Expand},N}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Expand},N}(\mathcal{B}_2) \\ & + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Expand},N}(\mathcal{B}_3) + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{HKDF.Expand},N}(\mathcal{B}_4)\end{aligned}$$

where $\mathcal{B}_1$ and $\mathcal{B}_3$ play against HKDF.Expand *for fixed output length l, and $\mathcal{B}_2$ and $\mathcal{B}_4$ play against* HKDF.Expand *for fixed output length d.*

This can be seen by a straightforward sequence of four games.

The main insight of this statement is that the bound is tight. Taking this together with the result of Corollary 2, we get that KDF is tightly secure if the underlying hash function of HKDF.Expand is modeled as the random oracle.

Corollary 3. *Let* KDF *be as defined in Fig. 3 and let* H *(i.e., the internal hash function used in* HKDF.Expand *(Sect. 2.8)) be modeled as a random oracle with output length* μ*. Then, for any adversary* $\mathcal{A}$

$$\mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{KDF},N}(\mathcal{A}) \leq 4 \cdot \left(\frac{N^2}{2^\kappa} + \frac{q_\mathsf{H}^2}{2^\mu} + \frac{2N}{2^\kappa} \right) = \frac{4 \cdot (N^2 + q_\mathsf{H}^2 + 2N)}{2^\mu}$$

where $\kappa = \mu$ *is the key length of the* HKDF.Expand *function used internally and* q_H *is the number of queries issued to the random oracle* H*.*

Remark 3. Note that the result on HKDF.Expand is independent of its outputs length. Also, although we use different output length HKDF.Expand instantiations in KDF, they still can share the same random oracle. This is due to the reason that the output length is determined by the number of rounds.

6. Tight MSKE-Security of the Full TLS 1.3 Handshake

Theorem 6. *Let* $\lambda, \mu \in \mathbb{N}$*, let* $\mathbb{G}$ *be a cyclic group of prime order* p *and* g *be a generator of prime order* q *subgroup of* $\mathbb{G}$*, let* SIG *be a digital signature scheme (Definition 6), and let* $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\mu$ *be a keyless hash function (Sect 2.5). Moreover, let* $\mathsf{KDF}\colon \{0,1\}^{2\mu} \times \{0,1\}^* \to \{0,1\}^\nu$*, let* $F_1, F_2\colon \mathbb{G} \times \{0,1\}^* \to \{0,1\}^{2\mu}$ *and let* $F_3, F_4\colon \mathbb{G} \times \{0,1\}^* \to \{0,1\}^\mu$ *be the functions defined in Figs. 3 and 2. We model* F_1*,* F_2*,* F_3 *and* F_4 *as random oracles.*

Further, let $\mathcal{A}$ *be an adversary against the* MSKE*-security (with key independence and stage-1 forward secrecy) of the TLS 1.3 handshake protocol as described in Sect. 4 running in time at most* t'*. Then, we can construct adversaries* $\mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_5, \ldots, \mathcal{B}_{10}$ *all running in time at most* t *such that* $t \approx t'$ *and*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MSKE}}_{\mathsf{TLS1.3}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{Coll-Res}}_{\mathsf{H}}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},|\mathcal{U}|}(\mathcal{B}_3) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_5) \\ & + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{KDF},n_s}(\mathcal{B}_6) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_7) + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{KDF},n_s}(\mathcal{B}_8) \\ & + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_9) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_{10}) + \frac{n_s^2}{2^\lambda}.\end{aligned}$$

where n_s *is the maximum number of sessions involved in TLS 1.3,* $\lambda = 256$ *is the nonce length defined in RFC 8446 [65] and* $\nu = 2l + 2d$*,*[11] *where* l *is the key length of the used AEAD scheme and* d *its IV length.*

Before we give the proof of Theorem 6, we want to plug in all results given in Sect. 5. In particular, we model the hash function H in addition as a random oracle and then we apply the results of Corollary 3 to Theorem 6. Also, we use that for a random oracle H the collision probability is given by $\mathsf{Adv}^{\mathsf{Coll-Res}}_{\mathsf{H}}(\mathcal{B}) \leq q_\mathsf{H}^2/2^\mu$ for any adversary $\mathcal{B}$, q_H being the number of queries issued to the random oracle, and μ being its output length.

[11] These quantities dependent on the selected ciphersuite. For details, see [65, Appx. B4].

Corollary 4. *Let* H *be modeled as a random oracle with output length* μ*. Apart from that let all other quantities be defined as in Theorem 6. Then,*

$$\mathsf{Adv}^{\mathsf{MSKE}}_{\mathrm{TLS1.3}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},\,|\mathcal{U}|}(\mathcal{B}_3) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},\,g}(\mathcal{B}_5) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},\,g}(\mathcal{B}_7) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},\,g}(\mathcal{B}_9) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},\,g}(\mathcal{B}_{10}) + \frac{8n_s^2 + 9q_{\mathsf{H}}^2 + 16n_s}{2^{\mu}} + \frac{n_s^2}{2^{256}}.$$

where q_{H} *is the number of queries to the random oracle* H*.*

Proof. Let break_δ be the event that the adversary $\mathcal{A}$ wins the Test-challenge in Game δ. Further, we write $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta] - \frac{1}{2}$.

Game 0 The initial game of the sequence is exactly the MSKE security experiment as given in Definition 10, i.e.,

$$\Pr[\mathsf{break}_0] = \frac{1}{2} + \mathsf{Adv}^{\mathsf{MSKE}}_{\mathrm{TLS1.3}}(\mathcal{A}) = \frac{1}{2} + \mathsf{Adv}_0.$$

Game 1 In this game, we want to make sure that there are no two honest sessions that sample the same nonce r_C, resp. r_S, in their `*Hello` message. Therefore, we raise the event $\mathsf{abort}_{\mathsf{nnc}}$ if session lbl samples nonce r that was sampled by a session $\mathsf{lbl}' \neq \mathsf{lbl}$ before.

The probability that such a collision among (at most) n_s independent samples from a set of size 2^λ occurs, is given by the birthday bound:

$$\Pr[\mathsf{abort}_{\mathsf{nnc}}] \leq n_s^2/2^\lambda.$$

Therefore,

$$\Pr[\mathsf{break}_0] \leq \Pr[\mathsf{break}_1] + n_s^2/2^\lambda \iff \mathsf{Adv}_0 \leq \mathsf{Adv}_1 + n_s^2/2^\lambda$$

where n_s is the maximum number of sessions involved in the protocol.

Game 2 In this game, we abort if there are two honest sessions that compute the same hash for different inputs in any evaluation of the hash function. We denote the corresponding event by $\mathsf{abort}_{\mathsf{hash}}$. Observe that if the challenger aborts the game due to this reason, we have found a hash collision in the hash function H. In order to bound $\Pr[\mathsf{abort}_{\mathsf{hash}}]$, we therefore construct an adversary $\mathcal{B}_2$ against the collision resistance (Definition 5) of H.

Construction of collision-finder $\mathcal{B}_2$*.* The reduction $\mathcal{B}_2$ simulates Game 1 for $\mathcal{A}$. If the challenger would raise the event $\mathsf{abort}_{\mathsf{hash}}$, $\mathcal{B}_2$ has found a collision and outputs the two (distinct) inputs to the hash function that resulted in the same hash as a collision.

Therefore, $\mathcal{B}_2$ wins if $\mathsf{abort}_{\mathsf{hash}}$ is raised in Game 2, i.e., $\Pr[\mathsf{abort}_{\mathsf{hash}}] \leq \mathsf{Adv}^{\mathsf{Coll-Res}}_{\mathsf{H}}(\mathcal{B}_2)$. Thus,

$$\mathsf{Adv}_1 \leq \mathsf{Adv}_2 + \mathsf{Adv}^{\mathsf{Coll-Res}}_{\mathsf{H}}(\mathcal{B}_2).$$

Game 3 In this game, we make sure that the adversary can only test sessions that have an honest (contributive) partner in the first stage, i.e., they agree on cid_1. To achieve this, we add another abort rule to the experiment. We raise the event $\mathsf{abort}_{\mathsf{SIG}}$ if the adversary tests a session that receives a signature valid under the public key pk_U of some party $U \in \mathcal{U}$ within a `ServerCertificateVerify` or `ClientCertificateVerify` message such that there is no honest and uncorrupted party that issued this signature. Here, we assume that the tested session in particular expects a signature, i.e., peer authentication is intended.

Let us analyze $\Pr[\mathsf{abort}_{\mathsf{SIG}}]$ first and then discuss the implications of this crucial step in detail afterward. Observe that if the challenger of the experiment aborts the game due to this rule, we found a forgery for the underlying signature scheme SIG. To that end, we can bound $\Pr[\mathsf{abort}_{\mathsf{SIG}}]$ by constructing an adversary $\mathcal{B}_3$ against the $\mathsf{MU-EUF-CMA}^{\mathsf{corr}}$-security of SIG for $|\mathcal{U}|$ users, where $\mathcal{U}$ is the set of users, running $\mathcal{A}$ as a subroutine.

Construction of forger $\mathcal{B}_3$ According to $\mathsf{Exp}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},\,|\mathcal{U}|}(\mathcal{B}_3)$ (Definition 8), adversary $\mathcal{B}_3$ receives $|\mathcal{U}|$ public keys $(pk_U)_{U\in\mathcal{U}}$[12] as input. Further, it has access to a signing oracle, access to a corruption oracle and outputs a forgery (U, m^*, σ). The forger $\mathcal{B}_3$ now simulates Game 3 for $\mathcal{A}$ except that it uses the public keys received by its challenger and the signing oracle to compute signatures of the respective user sessions. To be precise, a signature for user U is computed by issuing a signature query $(U, \cdot)$ to the signing oracle. Whenever, the adversary queries $\mathsf{Corrupt}(U)$, $\mathcal{B}_3$ relays the corruption query to its corruption oracle and sends the answer to the adversary. Now, if the challenger in Game 3 would raise the event $\mathsf{abort}_{\mathsf{SIG}}$, the forger has found a forgery. Namely, let $m^* = \ell_{\mathrm{SCV}} \,\|\, H_1$ (resp. $m^* = \ell_{\mathrm{CCV}} \,\|\, H_3$) (see Sect. 4), let σ be the signature involved in the event $\mathsf{abort}_{\mathsf{SIG}}$, which was received in the `ServerCertificateVerify` (resp. `ClientCertificateVerify`) message and let U be the party such that σ is valid under pk_U for message m^*, but no honest session output σ. The forger outputs (U, m^*, σ).

First, observe that $\mathcal{B}_3$ can use its oracles to perfectly simulate Game 3 for $\mathcal{A}$. Further, note that $\mathsf{abort}_{\mathsf{SIG}}$ only includes uncorrupted parties, which implies that U was never corrupted by both $\mathcal{A}$ and $\mathcal{B}_3$. Given this, $\mathcal{B}_3$ wins $\mathsf{Exp}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},\,|\mathcal{U}|}(\mathcal{B}_3)$ if $\mathsf{abort}_{\mathsf{SIG}}$ occurs. Formally,

$$\Pr[\mathsf{abort}_{\mathsf{SIG}}] \leq \mathsf{Adv}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},\,|(\mathcal{U}|}(\mathcal{B}_3).$$

[12] For convenience, assume that the public keys are indexed by the user identifier.

This gives us

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \mathsf{Adv}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},\,|\mathcal{U}|}(\mathcal{B}_3).$$

Next, we discuss the implications of this step. Suppose there is a tested session lbl that has no contributive first-stage partner, i.e., there is no $\mathsf{lbl}' \neq \mathsf{lbl}$ such that $\mathsf{lbl}.\mathsf{cid}_1 = \mathsf{lbl}'.\mathsf{cid}_1$. Due to the definition of the identifiers, we have that cid_1 is contained in sid_1 and thus lbl also has no session partner lbl' such that $\mathsf{lbl}.\mathsf{sid}_1 = \mathsf{lbl}'.\mathsf{sid}_1$. This in turn implies that lbl cannot have an honest (contributive) partner in any stage since $\mathsf{sid}_1 = \mathsf{cid}_2 = \mathsf{cid}_3 = \mathsf{cid}_4$ as described in Sect. 4.3. Recall that the model requires that keys can only be tested in unauthenticated stages if the tested session has a contributive partner (see Sect. 3.3, Test). Therefore, an adversary can only test a session in the absence of a contributive first-stage partner in stages 2–4 (stage 1 is always unauthenticated). Additionally, the adversary can only win in case of a client being tested if the key is responder-authenticated (i.e., $\mathsf{lbl}.\mathsf{auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$) and in case of a server the key is initiator-authenticated. This means whenever the adversary wants to test a session without a first-stage partner the messages `ServerCertificateVerify` or `ClientCertificateVerify` are sent. Recall that the expected signature is computed over $H_1 = \mathsf{H}(\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{CR}^*, \mathtt{SCRT})$ for `ServerCertificate Verify` resp. $H_3 = \mathsf{H}(\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{CR}^*, \mathtt{SCRT}, \mathtt{SF}, \mathtt{CCRT})$ for `Client CertificateVerify`. In particular, both contain $\mathsf{sid}_1 = \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}$ of the tested session lbl. However, as already mentioned above if the tested session has no contributive first-stage partner, it also has no session partner. Therefore, no honest session will seek to compute the expected signature. The only way to be able to test a session without a contributive partner therefore is to forge a signature. Due to the modifications of Game 2, we also made sure that the adversary cannot use collisions in the hash function. Observe that if a session accepts a `*CertificateVerfify` message it agrees on sid_1, and therefore cid_1, with the issuer of the respective signature. Hence, if the game does not abort due to the rule introduced in Game 3, the adversary is no longer able to issue a test query for a session that has no honest first-stage partner.

Game 4 In the previous game, we ensured that the adversary is only able to test a session when it has an honest (contributive) first-stage partner session added via a NewSession-query. This in particular implies that the adversary in this game is potentially allowed to issue Test-queries in each stage. In the following games, we make sure that from the view of the adversary the derived keys in each stage are uniformly distributed bit strings for every session. This game prepares these steps.

Intuitively, we embed randomizations of a DH tuple in the client and server sessions. In order to do that, we change the behavior of the challenger as follows: In the beginning of game the challenger chooses $a, b \xleftarrow{\$} \mathbb{Z}_p$ and computes the tuple $(A, B, C) := (g^a, g^b, g^{ab})$. Additionally, we change the implementation of the clients and servers.

– *Implementation of client sessions* Consider an arbitrary client session identified by lbl in SList. The challenger proceeds exactly as in Game 3 until the session chooses its key share. Instead of choosing a fresh exponent x as described in Sect. 4, it chooses a value $\tau_{\mathsf{lbl}} \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and outputs $X := A \cdot g^{\tau_{\mathsf{lbl}}} = g^{a+\tau_{\mathsf{lbl}}}$

as its key share in the `ClientKeyShare` message. Then, it proceeds as in Game 3 until it receives `ServerHello` and `ServerKeyShare`, and thus is able to compute the DHE key. If the value $Y = g^{b+\tau_{\mathsf{lbl}'}}$ received in `ServerKeyShare` was output by any honest server $\mathsf{lbl}' \in \mathsf{SList}$, we look up the used randomness $\tau_{\mathsf{lbl}'}$ and compute the DHE key as

$$Z := C \cdot A^{\tau_{\mathsf{lbl}'}} \cdot B^{\tau_{\mathsf{lbl}}} \cdot g^{\tau_{\mathsf{lbl}} \cdot \tau_{\mathsf{lbl}'}}.$$

Note that after a server session, which received lbl's `ClientKeyShare`, has sent (`SH`, `SKS`) it might not necessarily be received by lbl as it was sent. This is due to the fact that the signature is only sent after the first stage. Therefore, if lbl receives a `SKS` message that was not output by any honest server session, we do not know the corresponding value $\tau_{\mathsf{lbl}'}$ used in Y. Here, we cannot apply the above formula and use that we know the exponent a instead. In this case, we compute

$$Z := Y^{a+\tau_{\mathsf{lbl}}}.$$

The rest is exactly as in Game 3.

– *Implementation of server sessions* Consider an arbitrary server session identified by lbl' in SList. The challenger proceeds exactly as in Game 3 until the session chooses its key share. If lbl' receives (`CH`, `CKS`) output by an corrupted client lbl, then it continues proceeding as in Game 3. In particular, it chooses a fresh exponent y for its key share Y. However, if lbl is not corrupted, it chooses a value $\tau_{\mathsf{lbl}'} \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and outputs $Y := B \cdot g^{\tau_{\mathsf{lbl}'}} = g^{b+\tau_{\mathsf{lbl}'}}$. If the value $X = g^{a+\tau_{\mathsf{lbl}}}$ received in `CKS` was output by any client lbl, we look up the used randomness τ_{lbl} and compute the DHE key as

$$Z := C \cdot A^{\tau_{\mathsf{lbl}'}} \cdot B^{\tau_{\mathsf{lbl}}} \cdot g^{\tau_{\mathsf{lbl}} \cdot \tau_{\mathsf{lbl}'}}.$$

The rest is exactly as in Game 3

Although, we changed the way the sessions choose their key share values, we do not change their distribution. The key shares X, resp. Y, for all sessions still are distributed uniformly and independently at random in $\mathbb{G}$. Further, observe that the computation of Z yields valid DHE key based on the DH values sent by the respective parties. Thus, we have

$$\mathsf{Adv}_3 = \mathsf{Adv}_4.$$

Game 5 Using the preparations done in Game 4, we now argue that the adversary cannot learn anything about the values hts_C and hts_S unless it is able to break a variant of the CDH problem. To that end, we abort the game whenever the adversary queries the random oracle F_1 for (Z, `CH` ‖ `CKS` ‖ `SH` ‖ `SKS`), where (`CH`, `CKS`) and (`SH`, `SKS`) were output by honest sessions, respectively, and $Z = g^{xy}$ such that `CKS` contains g^x and `SKS` contains g^y. We denote the corresponding event by abort_{hts}. Observe that the challenger can efficiently check whether $Z = g^{xy}$ because it knows the exponents x

and y for every honest session. Note that if the challenger aborts the game due to this rule, the adversary was able to compute Z by only observing `ClientKeyShare` and `ServerKeyShare`. Thus, we can use $\mathcal{A}$ to break a computational DH problem. To be precise, we break the SDH problem in this case. This is due to the reason that the reduction needs the DDH oracle provided in SDH experiment given in Definition 3 to recognize a valid solution Z.

Construction of a SDH-adversary $\mathcal{B}_5$. The reduction $\mathcal{B}_5$ receives as input group elements $(A, B) = (g^a, g^b) \in \mathbb{G}^2$ and outputs a group element $C \in \mathbb{G}$. Moreover, it has access to a DDH oracle $\mathsf{DDH}(A, \cdot, \cdot)$. $\mathcal{B}_5$ simulates Game 4 for adversary $\mathcal{A}$, but instead of choosing fresh exponents in the beginning of the experiment, it uses (A, B) received as input. Further, the reduction does not know the exponents a and b. Therefore, it cannot compute Z for any session as described in Game 4. In these cases, we use that Z is input to the random oracle and the random oracle is under the control of the reduction. Whenever an honest session lbl would evaluate the random oracle F_1 to compute

$$hts_C \parallel hts_S := F_1(Z, \mathtt{CH} \parallel \mathtt{CKS} \parallel \mathtt{SH} \parallel \mathtt{SKS}),$$

$\mathcal{B}_5$ replaces Z by a place holder $\star_{\mathsf{lbl}}$, chooses a fresh image $v \xleftarrow{\$} \{0,1\}^{2\mu}$ and programs

$$(\star_{\mathsf{lbl}}, \mathtt{CH} \parallel \mathtt{CKS} \parallel \mathtt{SH} \parallel \mathtt{SKS}) \mapsto v$$

into the random oracle F_1. If lbl has a contributively partnered session lbl', we need to make sure that lbl' uses the same value v for its computations to ensure consistency.

Using the value $hts_C \parallel hts_S = v$, $\mathcal{B}_5$ can proceed with the computations without knowing a or b at all. Then, if the adversary queries the random oracle F_1 for $(Z, \mathtt{CH} \parallel \mathtt{CKS} \parallel \mathtt{SH} \parallel \mathtt{SKS})$, where $(\mathtt{CH}, \mathtt{CKS})$ was output by some honest client session lbl and $(\mathtt{SH}, \mathtt{SKS})$ was output by some honest server session lbl', $\mathcal{B}_5$ needs to check whether the adversary was able to break CDH, or whether it only made a random query. Therefore, it first needs to de-randomize Z. To that end, it looks up the randomnesses used by session lbl output $(\mathtt{CH}, \mathtt{CKS})$ and session lbl' output $(\mathtt{SH}, \mathtt{SKS})$, respectively. Note that due to the modifications of Game 1 these sessions are unique. We denote the respective randomnesses by τ_{lbl} and $\tau_{\mathsf{lbl}'}$, and compute

$$Z' := Z \cdot \left(A^{\tau_{\mathsf{lbl}'}} \cdot B^{\tau_{\mathsf{lbl}}} \cdot g^{\tau_{\mathsf{lbl}} \cdot \tau_{\mathsf{lbl}'}}\right)^{-1}.$$

Then, if $\mathsf{DDH}(B, Z') = 1$ it outputs Z' and halts. Otherwise, it continues to simulate the game for $\mathcal{A}$.

If the challenger aborts the game due to abort_{hts}, $\mathcal{B}_5$ would win with certainty. Thus, we have

$$\mathsf{Adv}_1 \leq \mathsf{Adv}_2 + \Pr[\mathsf{abort}_{hts}] \leq \mathsf{Adv}_2 + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_5).$$

This modification has an important implication. When the challenger never aborts due to abort_{hts}, we get that the adversary never gets hold of $hts_C \parallel hts_S$ computed by any

honest sessions not under its control. In particular, this means that from the view of the adversary $hts_C \parallel hts_S$ is a uniformly and independently (from the rest of the transcript) distributed bit string. Clearly, this also means that hts_C and hts_S considered individually are distributed uniformly and independently (also from each other!) at random.

The main technical step of our proof is the usage of function F_1. Using function F_1, we achieve that the DHE key Z and the messages `ClientHello` and `ServerHello` are bound together in a single function call (similar to the key derivation in TLS 1.2). If this would not be the case the reduction in this step would need to iterate over every pair of random values $(\tau_{\mathsf{lbl}}, \tau_{\mathsf{lbl}'})$, compute the value Z' for each of these pairs and check the DDH oracle for validity. This is rather inefficient (i.e., losing tightness) and is solved by the function F_1 in combination with Game 1 in an elegant way. Furthermore, we need F_1 to be a programmable random oracle to solve the "commitment problem" discussed by [38] to achieve tightness. The programming of the random oracle in combination with the SDH assumption is an alternative solution to the widely used PRF-ODH assumption used in TLS proofs.

Game 6 Observe that due to Game 3, the adversary is only able to issue Test-queries to sessions that have a first stage partner. In particular, this first stage partner is unique due to the modification given in Game 1. Since client and server sessions agree on cid_1 and thus on $\mathsf{sid}_1 = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS})$, where `CH` and `SH` are unique among the sessions, we also have a unique matching. In this game, we replace htk by a uniformly random value $\widetilde{htk} \xleftarrow{\$} \{0,1\}^\nu$ in all sessions that have a contributive first stage partner, i.e., in every session that can possibly be tested by the adversary. Note that this is already determined when the sessions compute htk. Clearly, we need to use the same value $\widetilde{htk}$ in the contributively partnered session. This modification should be unrecognizable unless the adversary is able to distinguish KDF from a truly random function. We analyze the difference in $\mathcal{A}$'s advantage introduced by this modification by constructing a reduction against the multi-user pseudorandomness (Definition 4) of KDF for n_s users, where n_s is the maximum number of sessions.

Construction of a MU-PRF-distinguisher $\mathcal{B}_6$. The reduction $\mathcal{B}_6$ has access to an oracle $\mathcal{O}_b(\cdot,\cdot)$ and outputs a bit $b' \in \{0,1\}$. For convenience, we assume that the users in experiment $\mathsf{Exp}^{\mathsf{MU\text{–}PRF}}_{\mathsf{KDF},n_s}\mathcal{B}_6$ are identified by the same value as the unique session labels used in the MSKE experiment $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$. $\mathcal{B}_6$ simulates Game 5 for $\mathcal{A}$. Whenever the challenger in Game 5 would compute htk, $\mathcal{B}_6$ first checks whether the session at hand, say lbl, has a contributive partner and if this is the case, it uses its oracle $\mathcal{O}_b(\mathsf{lbl}, \varepsilon)$ to compute htk. Then, it sets htk to the same value for the partnered session. For the non-partnered session as well as the rest of the simulation it proceeds exactly as in Game 5. Finally, if $\mathcal{B}_6$ outputs whatever $\mathcal{A}$ outputs.

If $\mathcal{O}_b$ is the KDF, then $\mathcal{B}_6$ perfectly simulates Game 5. Otherwise, if $\mathcal{O}_b$ is a truly random function it perfectly simulates Game 6. Thus,

$$\mathsf{Adv}_5 \leq \mathsf{Adv}_6 + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{KDF},n_s}(\mathcal{B}_6).$$

Now we have that the handshake traffic key *htk*, i.e., htk_C and htk_S, used to encrypt the handshake messages is distributed uniformly and independently at random in all sessions that can be tested. Hence, the probability to guess the challenge bit correctly after any $\mathsf{Test}(\mathsf{lbl}, 1)$-query is $1/2$. This is because the adversary is given a truly random key, independent of the test bit.

Game 7 Similar to Game 5, we next argue that the adversary does not learn anything about the values ats_C and ats_S unless it is able to break the SDH problem. We add a similar abort rule as before and abort the game whenever the adversary queries the random oracle F_2 for $(Z, \texttt{CH} \parallel \texttt{CKS} \parallel \texttt{SH} \parallel \texttt{SKS} \parallel \cdots \parallel \texttt{SF})$, where $(\texttt{CH}, \texttt{CKS})$ and $(\texttt{SH}, \texttt{SKS})$ were output by honest sessions, respectively, and $Z = g^{xy}$ such that $\texttt{CKS}$ contains g^x and $\texttt{SKS}$ contains g^y. We denote the corresponding event by abort_{ats}. To analyze the probability of abort_{ats}, we construct an adversary $\mathcal{B}_7$ against the SDH assumption. The construction of $\mathcal{B}_7$ follows along the lines of $\mathcal{B}_5$ except that we replace the considered random oracle by F_2 and the considered random oracle query by $(Z, \texttt{CH} \parallel \texttt{CKS} \parallel \texttt{SH} \parallel \texttt{SKS} \parallel \cdots \parallel \texttt{SF})$.

Following the same argumentation as in Game 5, we get

$$\mathsf{Adv}_6 \leq \mathsf{Adv}_7 + \Pr[\mathsf{abort}_{ats}] \leq \mathsf{Adv}_7 + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},\, g}(\mathcal{B}_7).$$

Game 8 In this game, we replace *atk* by a uniformly random value $\widetilde{atk} \xleftarrow{\$} \{0,1\}^{\nu}$ in all sessions that have a contributive first stage partner, i.e., in every session that possibly can be tested by the adversary. Note that this is already determined when the sessions compute *atk*. Clearly, we need to use the same value $\widetilde{atk}$ in the contributively partnered session. This step is analogous to Game 6, and thus supplies

$$\mathsf{Adv}_7 \leq \mathsf{Adv}_8 + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{KDF}, n_s}(\mathcal{B}_8).$$

Game 9 In this game, we argue that the adversary does not learn anything about the stage-3 key *ems*. To that end, we employ the same argumentation already given in Games 5 and 7 that this is the case unless the adversary is not able to break the SDH assumption. Formally, we abort the game whenever the adversary queries the random oracle F_3 for $(Z, \texttt{CH} \parallel \texttt{CKS} \parallel \texttt{SH} \parallel \texttt{SKS} \parallel \cdots \parallel \texttt{SF})$, where $(\texttt{CH}, \texttt{CKS})$ and $(\texttt{SH}, \texttt{SKS})$ were output by honest sessions, respectively, and $Z = g^{xy}$ such that $\texttt{CKS}$ contains g^x and $\texttt{SKS}$ contains g^y. We denote the corresponding event by abort_{ems}. To analyze the probability of abort_{ems}, we construct an adversary $\mathcal{B}_9$ against the SDH assumption. The construction of $\mathcal{B}_9$ follows along the lines of $\mathcal{B}_5$ except that we replace the considered random oracle by F_3 and the considered random oracle query by $(Z, \texttt{CH} \parallel \texttt{CKS} \parallel \texttt{SH} \parallel \texttt{SKS} \parallel \cdots \parallel \texttt{SF})$.

Following the same argumentation as in Games 5 and 7, we get

$$\mathsf{Adv}_8 \leq \mathsf{Adv}_9 + \Pr[\mathsf{abort}_{ems}] \leq \mathsf{Adv}_9 + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},\, g}(\mathcal{B}_9).$$

Game 10 In this game, we argue that the adversary does not learn anything about the stage-4 key *rms*. To that end, we employ the same argumentation already given in Games 5, 7 and 9 that this is the case unless the adversary is not able to break the SDH assumption. Formally, we abort the game whenever the adversary queries the random oracle F_4 for

$(Z, \texttt{CH}\|\texttt{CKS}\|\texttt{SH}\|\texttt{SKS}\|\cdots\|\texttt{CF})$, where $(\texttt{CH}, \texttt{CKS})$ and $(\texttt{SH}, \texttt{SKS})$ were output by honest sessions, respectively, and $Z = g^{xy}$ such that $\texttt{CKS}$ contains g^x and $\texttt{SKS}$ contains g^y. We denote the corresponding event by abort_{rms}. To analyze the probability of abort_{rms}, we construct an adversary $\mathcal{B}_{10}$ against the SDH assumption. The construction of $\mathcal{B}_{10}$ follows along the lines of $\mathcal{B}_5$ except that we replace the considered random oracle by F_4 and the considered random oracle query by $(Z, \texttt{CH} \,\|\, \texttt{CKS} \,\|\, \texttt{SH} \,\|\, \texttt{SKS} \,\|\, \cdots \,\|\, \texttt{CF})$.

Following the same argumentation as in Games 5, 7 and 9, we get

$$\mathsf{Adv}_9 \leq \mathsf{Adv}_{10} + \Pr[\mathsf{abort}_{rms}] \leq \mathsf{Adv}_{10} + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_{10}).$$

Closing remarks Finally, we have the case that in Game 10 the keys of either stage, i.e., *htk*, *atk*, *ems* and *rms*, are from the view of the adversary uniformly and independently distributed bit strings. This implies that for each Test-query, the distribution of the received key is independent of the test bit b_{Test}. Therefore, the probability that the adversary guesses the test bit correctly is $1/2$. Hence, the advantage in guessing the test bit correctly in Game 10 is

$$\mathsf{Adv}_{10} = 0.$$

Overall, we get

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MSKE}}_{\mathrm{TLS1.3}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{Coll-Res}}_{\mathsf{H}}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{MU-EUF-CMA}^{\mathsf{corr}}}_{\mathsf{SIG},|\mathcal{U}|}(\mathcal{B}_3) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_5) \\ & + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{KDF},n_s}(\mathcal{B}_6) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_7) + \mathsf{Adv}^{\mathsf{MU-PRF}}_{\mathsf{KDF},n_s}(\mathcal{B}_8) \\ & + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_9) + \mathsf{Adv}^{\mathsf{SDH}}_{\mathbb{G},g}(\mathcal{B}_{10}) + \frac{n_s^2}{2^\lambda}.\end{aligned}$$

7. Tight Compositional Security of MSKE Protocols

In this section, we revisit the latest version of the composition theorem of MSKE-protocols stated by [40, Thm. 4.4]. The original theorem suffers from a linear loss in the number of sessions involved in the protocol. However, Günther already conjectures that the hybrid argument inducing the linear loss in the proof of the composition theorem might be removed by making use of the multiple Test-queries allowed in MSKE model (Sect. 3). Currently, each hybrid step only uses a single Test-query in the reduction to the MSKE-security (see [40, Lem. 4.5]). Using as many Test-queries in the reduction to the MSKE experiment as number of sessions removes the necessity of a hybrid argument.

In the following, we recall necessary preliminaries from [40], state the improved (in terms of tightness) composition theorem, prove it and finally discuss the implications on TLS 1.3.

7.1. *Compositional Security of MSKE Protocols*

In general, key exchange protocols are only reasonable when used in combination with another protocol using the derived keys. However, it is not trivially clear that the composition of a key exchange protocol with a "partner protocol" still remains secure. To tame complexity of the security analysis it is always desirable to be as modular as possible. Therefore, the appealing option is to prove the two protocols secure on their own and apply some generic composition theorem to show that the composition remains secure. In case of the standard Bellare–Rogaway key exchange model [12], [23] were able to show that a protocol that is secure in the Bellare-Rogaway model can be securely composed with an arbitrary symmetric key protocol.

Fischlin et al. [36] transferred this result into the multi-stage setting, which also evolved over time with the MSKE model (Sect. 3), see [31,33,37,40]. We refer to the most recent version presented in [40]. The result states that an MSKE-secure protocol providing key-independence, stage-j forward secrecy and multi-stage session matching (see below), can be securely composed with an arbitrary symmetric key-protocol at a forward-secure, external and non-replayable stage.

7.1.1. *Composed Security Experiment*

Next, we describe the composed experiment. Let $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}$ be the MSKE security experiment involving protocol KE and let $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}$ be some security experiment for an arbitrary symmetric key protocol Π. As [40], we fix some stage i and consider only the keys of this stage to be used in Π.

We denote the composition of KE and Π, where only the stage-i keys are used in Π, by $\mathsf{KE}_i; \Pi$. The composed security experiment is denoted by $\mathsf{Exp}^{\mathsf{MSKE;GOAL}}_{\mathsf{KE}_i;\Pi}$ and is defined as follows for some adversary $\mathcal{A}$: In essence, adversary $\mathcal{A}$ is challenged in the composed game to win the sub-experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$ with the modification of keys being not chosen by the challenger but originate from KE. To that end, we start by simulating $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$. We introduce a special key registration query in $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}$ to allow using keys originating from KE. This query is only executed by the composed game. Upon acceptance of a session key key_i in stage i of some session of KE such that the key was derived by either an authenticated session or a contributively partnered session, we register key_i as a new key in the sub-experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$ allowing the adversary to issue queries as defined in $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}$ involving this (and possibly previous registered) key(s). To be precise, we only register accepted keys key_i of a session lbl such that either lbl has an *authenticated* communication partner, i.e., $\mathsf{lbl.auth}_i = \mathsf{mutual}$ or $\mathsf{lbl.auth}_i = \mathsf{unilateral}$ and $\mathsf{lbl.role} = \mathsf{initiator}$, or has an honest contributive partner, i.e., there is a session lbl' in KE with $\mathsf{lbl.cid}_i = \mathsf{lbl}'.\mathsf{cid}_i$. This is important to make sure that adversary has no information about the registered keys. Otherwise, we cannot guarantee security in $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$. The composed game can be viewed as a concurrent execution of $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$ and $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$ by some challenger. The adversary is allowed to issue queries of both experiment, which the challenger essentially just relays to the suitable sub-experiment. In case of queries, we need to make the following restrictions: First, as mentioned above, the key registration query for $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}$ can only be asked by the composed experiment challenger. Also, this query is only allowed to be executed when a

key is accepted in stage i. Second, Reveal-queries of $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}$ are not allowed for stage i keys. This is due to the reason that the corruption of stage i keys might only be captured by $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}$. Third, the Test-query makes no sense in the composed experiment as we aim for the security goals of Π in the composed experiment. Finally, adversary $\mathcal{A}$ wins the composed experiment, if it is able to win the sub-experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$ with access to the oracles discussed above.

7.1.2. *Multi-stage Session Matching*

We recall the definition of a *multi-stage session matching algorithm* stated by [40].

Definition 11. Let $\mathcal{A}$ be any adversary interacting in the experiment $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$. We say an algorithm $\mathcal{M}$ is a *multi-stage session matching algorithm* if the following holds. On input a stage i, the public information of the experiment, an ordered list of all queries made by $\mathcal{A}$ and responses from $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$ at any point of the experiment execution, as well as a list of all stage-j keys with $j < i$ for all session accepted at this point, algorithm $\mathcal{M}$ outputs two lists of pairs of all session in stage i. Here, the first list contains exactly those pairs of sessions that are partnered (i.e., they share the same session identifier sid_i). The second list contains exactly those pairs of sessions that are contributively partnered (i.e., they share the same contributive identifier cid_i).

We say KE allows for multi-stage session matching, if such an algorithm $\mathcal{M}$ exists for KE.

Note that the session matching algorithms can be run at any point of the execution of the key exchange protocol.

7.2. *Improved Multi-stage Compositional Security*

Next, we restate the composition theorem of [40] with improved tightness. Compared to [40], we reduced the factor of the MSKE advantage to 1. This was already conjectured in [40].

Theorem 7. *Let* KE *be a* MSKE*-secure (Definition 10) key exchange protocol with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE})$ *and key length* ν *providing key independence and stage-*j *forward secrecy and that allows for multi-stage session matching (Definition 11). Let* Π *be a symmetric-key protocol that is secure in the sense of some security experiment* $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}$ *and keys are chosen uniformly at random from* $\{0,1\}^{\nu}$*. Further, let* $i \in \mathsf{M}$ *with* $i \geq j$ *be a stage of* KE *such that* $\mathsf{USE}_i = \mathsf{external}$*. Then, for any adversary* $\mathcal{A}$ *running in* $\mathsf{Exp}^{\mathsf{MSKE};\mathsf{GOAL}}_{\mathsf{KE}_i;\Pi}(\mathcal{A})$*, we can construct adversaries* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *such that*

$$\mathsf{Adv}^{\mathsf{MSKE};\mathsf{GOAL}}_{\mathsf{KE}_i;\Pi}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3).$$

The proof of Theorem 7 basically follows along the lines of [40, Thm. 4.4].

Proof. The proof follows a sequence of games [68]. Let Adv_{δ} denote the advantage of the adversary $\mathcal{A}$ in game δ.

Game 0 The first game equals the composed security experiment $\mathsf{Exp}^{\mathsf{MSKE;GOAL}}_{\mathsf{KE}_i;\Pi}(\mathcal{A})$, i.e.,

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{MSKE;GOAL}}_{\mathsf{KE}_i;\Pi}(\mathcal{A}).$$

Game 1 Next, we modify Game 0 such that it always outputs the same key key_i for two partnered session in stage i. To that end, we raise the event $\mathsf{abort}_{\mathsf{Match}}$ in case two partnered session do not agree on the same key. Therefore,

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \Pr[\mathsf{abort}_{\mathsf{Match}}].$$

To analyze $\Pr[\mathsf{abort}_{\mathsf{Match}}]$, we construct an adversary $\mathcal{B}_1$ against the Match-security of KE running $\mathcal{A}$ as a subroutine.

Construction of adversary $\mathcal{B}_1$. The adversary $\mathcal{B}_1$ simply relays every query made by $\mathcal{A}$ in the sub-experiment $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$ to its challenger and sends the responses back to $\mathcal{A}$. The second sub-experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}$ can simulate $\mathcal{B}$ by itself. By doing this, $\mathcal{B}_1$ provides a perfect simulation of the composed security experiment for $\mathcal{A}$. Thus, if $\mathcal{B}_1$ is run with an adversary $\mathcal{A}$ that triggers the event $\mathsf{abort}_{\mathsf{Match}}$ it will always succeed in breaking the Match-security of KE, i.e.,

$$\Pr[\mathsf{abort}_{\mathsf{Match}}] \leq \mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE}}(\mathcal{B}_1).$$

Game 2 In this game, we make the crucial step of this proof. This is also the step that differs from the analysis of [40]. We change the way how the keys registered in the sub-experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}$ are chosen. That is, whenever a session accepts in stage i such that the accepted key would be registered in the sub-experiment (i.e., either peer-authentication or an honestly partnered session, for details see above), we do not register the real key key_i but register a freshly chosen key $\mathsf{key}'_i \xleftarrow{\$} \{0,1\}^{\nu}$.

We claim that based on adversary $\mathcal{A}$, we can construct an $\mathcal{B}_2$ such that

$$|\mathsf{Adv}_1 - \mathsf{Adv}_2| \leq \mathsf{Adv}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{B}_2).$$

Note that Adv_1 and Adv_2 depend on $\mathcal{A}$.

Construction of $\mathcal{B}_2$. The adversary $\mathcal{B}_2$ simulates the composed security experiment $\mathsf{Exp}^{\mathsf{MSKE;GOAL}}_{\mathsf{KE}_i;\Pi}(\mathcal{A})$ for $\mathcal{A}$. To that end, it relays all queries of $\mathcal{A}$ issued to the MSKE sub-experiment directly to its MSKE-experiment, with a few exceptions described below. In case of the symmetric-key sub-experiment, $\mathcal{B}_2$ is able to simulate it on its own. Important to note it that it needs to use the stage-i keys established during the key exchange. To describe the simulation of the oracles in detail, we use the following two maps:

1. $\mathsf{SDATA}\colon \mathsf{LABELS} \to \{\mathsf{initiator}, \mathsf{responder}\} \times \{\mathsf{unauth}, \mathsf{mutual}, \mathsf{unilateral}\} \times (\{0,1\}^{\nu})^{i-1}$, which stores the role, the authentication type used in stage i and all sessions keys for stages $j < i$ of each session involved in KE, and

2. $\mathsf{SKEY}\colon \mathsf{LABELS} \to \{0,1\}^{\nu}$, which stores the stage-i key that is registered in the symmetric key protocol.

For the oracles, we have the following:

- NewSession, Reveal and Corrupt queries are simply forwarded to $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{B}_2)$ and responses directly forwarded to $\mathcal{A}$. In addition, $\mathcal{B}_2$ adds after every NewSession-query an entry to SDATA containing the issued label, the respective role and the respective authentication type for stage i.
  Let us argue why it is valid to just forward Reveal and Corrupt to the experiment $\mathcal{B}_2$ is running in. In case of Reveal, note that we assume key independence for KE. This means that any query $\mathsf{Reveal}(\cdot, i')$ with $i' \neq i$ ($\mathsf{Reveal}(\cdot, i)$ is forbidden in the composed experiment) does not affect the security of a stage-i key (since they are independent). In case of Corrupt, the stage-j ($j \leq i$) forward secrecy assumed for KE make sure that stage i keys are untouched by Corrupt queries.[13]
- Send-queries are also forwarded to the experiment $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{B}_2)$ and answered relayed back to $\mathcal{A}$. However, $\mathcal{B}_2$ is doing the following. If some session lbl changes in $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{B}_2)$ to an accepting state for some stage $j < i$ in response to a Send-query, adversary $\mathcal{B}_2$ issues a $\mathsf{Reveal}(\mathsf{lbl}, j)$-query and stores the key key_j it gets in response in the map SDATA. Due to the fact that KE is key independent by assumption, such a Reveal query does not affect any stage-i key. We need to get these keys to run the multi-stage session matching algorithm in next step.
  When any session lbl changes into an accepting state in stage i, $\mathcal{B}_2$ runs the multi-stage session matching algorithm (Definition 11) for KE on input all queries and respective responses $\mathcal{A}$ issued to the sub-experiment $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{A})$ of the composed experiment as well as all established stage-j keys for $j < i$ stored in SDATA. The algorithm outputs all sessions that are partnered and also those that are contributively partnered in stage i.
  If some session lbl is partnered with some other session lbl' such that $\mathsf{SKEY}(\mathsf{lbl}')$ is set, $\mathcal{B}_2$ sets $\mathsf{SKEY}(\mathsf{lbl})$ accordingly and provides $\mathcal{A}$ with an identifier for $\mathsf{SKEY}(\mathsf{lbl})$ of lbl in the sub-experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$ to enable queries on the key. Due to Game 1, we have already ensured that these keys are always identical.
  If this is not the case, $\mathcal{B}_2$ needs to check whether the key of lbl is supposed to be registered in the sub-experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$. Therefore, it checks whether lbl either lbl has an authenticated communication partner, i.e., $\mathsf{lbl}.\mathsf{auth}_i = \mathsf{mutual}$ or $\mathsf{lbl}.\mathsf{auth}_i = \mathsf{unilateral}$ and $\mathsf{lbl}.\mathsf{role} = \mathsf{initiator}$ (obtained using SDATA), or has an honest contributive partner, i.e., there is a session lbl' in KE with $\mathsf{lbl}.\mathsf{cid}_i = \mathsf{lbl}'.\mathsf{cid}_i$. In case this is true, $\mathcal{B}_2$ queries $\mathsf{Test}(\mathsf{lbl}, i)$, stores the resulting key in $\mathsf{SKEY}(\mathsf{lbl})$ and registers $\mathsf{SKEY}(\mathsf{lbl})$ for lbl in the sub-experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$. The adversary is given an identifier for $\mathsf{SKEY}(\mathsf{lbl})$ to query issues on that key.

[13] Note that compared to [40], we do not mention the non-replayability of stage i at this point. This is due to the reason that we do not have RevealSemiStaticKey-queries in our model. However, non-replayability is important for a two-party symmetric key protocol to ensure that a key is shared only between two parties.

It remains to argue that $\mathcal{B}_2$ wins the experiment $\mathsf{Exp}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{B}_2)$ if $\mathcal{A}$ wins the composed experiment $\mathsf{Exp}^{\mathsf{MSKE;GOAL}}_{\mathsf{KE}_i;\Pi}(\mathcal{A})$. To that end, we argue that 1) $\mathcal{B}_2$ never triggers the lost-flag to be set and 2) $\mathcal{B}_2$ provides a perfect simulation for $\mathcal{A}$.

First, checking for a partnered session that might already have a set value for SKEY, ensures that $\mathcal{B}_2$ only tests the session of two partnered sessions that accepts the key first (see Sect. 3.3, Test). Moreover, it never both tests and reveals a key (see Definition 10, 4.). This is due to the reason that $\mathcal{B}_2$ only tests stage i keys, never reveals stage i keys and the adversary is not allowed to query the Test-oracle as well as $\mathsf{Reveal}(\cdot, i)$-queries at all. Finally, as we only register keys of sessions use authentication or have a contributive partner, we also never test a session with unauthenticated or dishonest contributive partner (see Sect. 3.3, Test). Hence, $\mathcal{B}_2$ never triggers the lost-flag to be set.

Second, we argue that $\mathcal{B}_2$ provides a perfect simulation for $\mathcal{A}$. Note that stage i of KE is external, i.e., all keys established are external keys, if the Test-query returns a random key (i.e., $b_{\mathsf{Test}} = 0$) then it does replace the actual key established in the protocol, which does not corrupt a perfect simulation. First of all, $\mathcal{B}_2$ outputs 1 if $\mathcal{A}$ terminates and wins the composed security experiment $\mathsf{Exp}^{\mathsf{MSKE;GOAL}}_{\mathsf{KE}_i;\Pi}(\mathcal{A})$. If $\mathcal{A}$ looses, it outputs 0. Hence, $\mathcal{B}_2$ perfectly simulates in case $b_{\mathsf{Test}} = 0$, Game 2 and in case $b_{\mathsf{Test}} = 1$, perfectly Game 1. Therefore, we have that $\mathcal{B}_2$ always wins if $\mathcal{A}$ wins and

$$|\mathsf{Adv}_1 - \mathsf{Adv}_2| \leq \mathsf{Adv}^{\mathsf{MSKE}}_{\mathsf{KE}}(\mathcal{B}_2).$$

Final Step Finally, we informally claim that based on an adversary $\mathcal{A}$ winning in Game 2, we can construct an adversary $\mathcal{B}_3$ that wins the game $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3)$ against Π. More formally, we show that given $\mathcal{A}$ running in Game 2, we can construct an adversary $\mathcal{B}_3$ running in $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3)$ such that

$$\mathsf{Adv}_2 \leq \mathsf{Adv}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3).$$

Construction of $\mathcal{B}_3$ The adversary $\mathcal{B}_3$ simulates Game 2 for $\mathcal{A}$. Here, it simulates the key exchange sub-experiment completely on its own while forwarding any query issued to symmetric key sub-experiment to its experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3)$. Due to the reason that stage-i keys are external, they are independent of the actual protocol, i.e., $\mathcal{B}_3$ can provide a perfect simulation on its own, since the keys are not effecting the protocol. However, we need to be careful when registering the keys in the experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3)$. Therefore, we distinguish two cases when a session accepts a key in response to a Send-query:

1. If the session accepting the key is partnered, $\mathcal{B}_3$ needs to make sure that the same key is registered for the partnered session in $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3)$ and therefore returns the respective identifier of that key to the adversary. This is important to enable the adversary to issue queries in the symmetric key experiment.
2. Otherwise, $\mathcal{B}_3$ asks its experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3)$ for a fresh key, which it then registers this fresh key in the composed experiment for the according session.

Due to Game 2 all keys are distributed uniformly at random (independent from each other). This is the same way the keys are chosen in $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3)$. In addition, ensuring consistency between partnered sessions provides us that $\mathcal{B}_3$ perfectly simulates Game 2

for $\mathcal{A}$. When $\mathcal{A}$ stops, $\mathcal{B}_3$ simply outputs whatever $\mathcal{A}$ outputs. Recall that $\mathcal{A}$ in the composed security experiment essentially is challenged to win the experiment $\mathsf{Exp}^{\mathsf{GOAL}}_{\Pi}(\mathcal{A})$ with keys originating from the key exchange protocol. Therefore, $\mathcal{B}_3$ will provide a correctly formed output. Hence, we obtain the following relation as claimed before:

$$\mathsf{Adv}_2 \leq \mathsf{Adv}^{\mathsf{GOAL}}_{\Pi}(\mathcal{B}_3).$$

Taking all steps together, we get the statement claimed in Theorem 7.

7.3. *Tight Compositional Security of TLS 1.3*

The results on Match-security (Theorem 1) and on MSKE-security (Theorem 6) show that the tight security of TLS 1.3 is in principle achievable given the TLS 1.3 Record Protocol provides tight security as well. [13] and [41] investigated the multi-user security of AES-GCM as used in TLS 1.3. In particular, Hoang et al. were able to give a tight bound for the *nonce-randomization mechanism* ([41, Thm. 4.2]) that is adopted in TLS 1.3 ([65, Sect. 5.3]).

7.3.1. *Using the Bound of Hoang et al. for the TLS 1.3 Record Protocol*

To formally apply the result of [41] in Theorem 7, we need to introduce a trivial intermediate step. TLS 1.3 as defined in Sect. 4 derives every traffic (i.e., handshake and application traffic key) as a pair of client and server key. To reflect this in the symmetric key protocol, we define an intermediate experiment that takes a key that can be split up into two separate keys and uses these keys for an authenticated encryption experiment. The adversary then can issue queries for both client and server individually.

To incorporate the result of [41], we extend their authenticated encryption experiment (see [41, Sect. 2.1]) accordingly.

Definition 12. Let $\mathsf{AEAD}[E] = (\mathsf{AEAD}[E].\mathsf{Gen}, \mathsf{AEAD}[E].\mathsf{Enc}, \mathsf{AEAD}[E].\mathsf{Dec})$ be an authenticated encryption scheme with associated data using an ideal cipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, where $k, n \in \mathbb{N}$. Further, let the Gen output keys of length $\nu \in \mathbb{N}$ such that $2 \mid \nu$ and Enc outputs ciphertexts that are λ-bit longer then the message.

Consider the following experiment $\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A})$ played between a challenger and an adversary $\mathcal{A}$:

1. The challenger generates a key $K_i \xleftarrow{\$} \mathsf{AEAD}[E].\mathsf{Gen}$ for each user $i \in [U]$. We view each key as $K_i^C \parallel K_i^S := K_i$. Further, it samples a bit $b \xleftarrow{\$} \{0,1\}$.
2. The adversary may adaptively issue queries to the following oracles:
   - $\mathsf{ClientEnc}(i, N, A, M)$: If $b = 0$, output $\mathsf{AEAD}[E].\mathsf{Enc}(K_i^C, N, A, M)$. Otherwise, return $C \xleftarrow{\$} \{0,1\}^{|M|+\lambda}$.
   - $\mathsf{ClientVf}(i, N, A, C)$: If $b = 0$, set $V := \mathsf{AEAD}[E].\mathsf{Dec}(K_i^C, N, A, C)$ and return $(V \neq \perp)$. Otherwise, return false.
   - $\mathsf{ServerEnc}(i, N, A, M)$: If $b = 0$, output $\mathsf{AEAD}[E].\mathsf{Enc}(K_i^S, N, A, M)$. Otherwise, return $C \xleftarrow{\$} \{0,1\}^{|M|+\lambda}$.

- $\mathsf{ServerVf}(i, N, A, C)$: If $b = 0$, set $V := \mathsf{AEAD}[E].\mathsf{Dec}(K_i^S, N, A, C)$ and return $(V \neq \perp)$. Otherwise, return false.
- $\mathsf{Prim}(J, X)$: If $X = (+, x)$, return $E_J(x)$. Otherwise, if $X = (-, y)$, return $E_j^{-1}(y)$.

Further, we restrict the adversary in the following sense: We require $\mathcal{A}$ that it must not repeat (i, N) for any $\mathsf{ClientEnc}$ and $\mathsf{ServerEnc}$ query, respectively. It can repeat nonces for $\mathsf{ClientVf}$ and $\mathsf{ServerVf}$ queries. However, it is not allowed to issue a query $\{\mathsf{Client}, \mathsf{Server}\}\mathsf{Enc}(i, N, A, M)$ to obtain a ciphertext C and then query the corresponding $\{\mathsf{Client}, \mathsf{Server}\}\mathsf{Vf}(i, N, A, C)$ oracle to avoid trivial wins.

3. Finally, the adversary outputs a guess $b' \in \{0, 1\}$.
4. The experiment outputs 1 if $b = b'$ and the above stated conditions were not hurt. Otherwise, it outputs 0.

We denote the advantage of an adversary $\mathcal{A}$ in winning the experiment $\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A})$ by

$$\mathsf{Adv}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A}) := \left| \Pr[\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A})] - \frac{1}{2} \right|$$

where $\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A})$ denotes the experiment defined above.

It remains to show that the tight bound given by [41] implies tight security of AES-GCM in the model given in Definition 12.

Theorem 8. *Let* $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$*, where* $k, n \in \mathbb{N}$*, be a blockcipher that is modeled as an ideal cipher. Let* $\mathsf{AEAD}[E] = (\mathsf{AEAD}[E].\mathsf{Gen}, \mathsf{AEAD}[E].\mathsf{Enc}, \mathsf{AEAD}[E].\mathsf{Dec})$ *be an authenticated encryption scheme with associated data using an ideal cipher* $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$*, where* $k, n \in \mathbb{N}$*. Further, let the* Gen *output keys of length* $v \in \mathbb{N}$ *such that* $2 \mid v$ *and* Enc *outputs ciphertexts that are* λ*-bit longer then the message.*

Further, let $\mathcal{A}$ *be an adversary against* $\mathsf{AEAD}[E]$ *in* $\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A})$*. Then, we can construct an adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A}) \leq \frac{1}{2}\,\mathsf{Adv}^{\mathrm{mu-ae}}_{\mathsf{AEAD}[E],2U}(\mathcal{B})$$

where $\mathsf{Adv}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\cdot)$ *is defined as in Definition* 12 *and* $\mathsf{Adv}^{\mathrm{mu-ae}}_{\mathsf{AEAD}[E],2U}(\cdot)$ *is as defined in* [41, Sect. 2.1].

Remark 4. Observe that the factor of $\frac{1}{2}$ results from the fact that the experiment $\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\cdot)$ is a "bit-guessing experiment" in contrast to the "real-or-random experiment" used in [41]. Further, we add a parameter of users to the advantage notation of $\mathsf{Adv}^{\mathrm{mu-ae}}_{\mathsf{AEAD}[E],2U}(\cdot)$ to clarify the number of users running in the corresponding experiment.

Proof (Sketch). We briefly give the main idea in form of sketch. The adversary $\mathcal{B}$ gets keys $(K_1, \ldots, K_{2U})$ as input from its challenger. It then defines U keys

$$(K'_{(1,2)}, K'_{(3,4)}, \ldots, K'_{(2U-1,2U)}) := (K_1 \parallel K_2, K_3 \parallel K_4, \ldots, K_{2U-1} \parallel K_{2U}).$$

Subsequently, it hands $(K'_{(1,2)}, K'_{(3,4)}, \ldots, K'_{(2U-1,2U)})$ to $\mathcal{A}$ and simulates the experiment $\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A})$. The important step now is the correct mapping of oracles:

Let (i, j) be any key identifier such that $K_{(i,j)}$ was handed to $\mathcal{A}$. We map the oracles of $\mathcal{A}$ to the oracles of $\mathcal{B}$ as follows:

$$\begin{aligned} \mathsf{ClientEnc}((i,j),\cdot) &\mapsto \mathsf{Enc}(i,\cdot) & \mathsf{ClientVf}((i,j),\cdot) &\mapsto \mathsf{Vf}(i,\cdot) \\ \mathsf{ServerEnc}((i,j),\cdot) &\mapsto \mathsf{Enc}(j,\cdot) & \mathsf{ServerVf}((i,j),\cdot) &\mapsto \mathsf{Vf}(j,\cdot) \end{aligned}$$

The Prim oracle queries are just relayed to the Prim oracle of $\mathcal{B}$. This way $\mathcal{B}$ provides a perfect simulation of $\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\mathcal{A})$.

7.3.2. *Implications of Theorem 8*

We stress that the tight bound given for the nonce-randomization of AES-GCM given in [41] is independent of the number of users. Thus, Theorem 8 shows that the given tight bound still applies to the experiment $\mathsf{Exp}^{\mathsf{COMP-MU-AE}}_{\mathsf{AEAD}[E],U}(\cdot)$ although it involves twice as much users.

Using the composition theorem stated in Theorem 7 we obtain that given a MSKE protocol KE that provides a tight bound on both Match-security (Definition 9) and MSKE-security (Definition 10) and a symmetric key protocol Π that provides a tight bound in its respective security model, we result in a tightly secure composed protocol. For TLS 1.3, we were able to show in Theorem 6 when instantiated with suitable, tightly secure primitives that the TLS 1.3 Handshake protocol is tightly MSKE-secure in the random oracle model. The results of Sect. 5 on the tight security of the PRFs used in the TLS 1.3 Handshake protocol supplies in Corollary 4 that we are indeed able to instantiate TLS 1.3 tightly secure in the random oracle model with the exception of the signature scheme SIG. For the TLS 1.3 Record Layer, we can use Theorem 8 and the intermediate experiment defined in Definition 12 to incorporate the tight bound of [41] for AES-GCM's nonce randomization mechanism. Hence, we have that the composition of the TLS 1.3 Handshake protocol when using the keys derived in stage 2, i.e., the application traffic key, in the experiment Definition 12 enables an *almost* tight instantiation of TLS 1.3 for every `TLS_AES_*_GCM_*` ciphersuites.

Also, security for the Record Protocol in a stronger security model in a tightly secure manner remains an open question. The result of [41] is stateless, however as attacks in the past have shown, security against reordering attacks is an important property for TLS. We note at this point that the in the way we did it in Definition 12, one can transfer every multi-user notion into a similar experiment. Therefore, stronger results easily can be incorporated by a generalization of the experiment given in Definition 12.

Acknowledgements

We thank the anonymous reviewers for their extensive and valuable comments that helped to improve the presentation of the paper a lot.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

[1] M. Abdalla, M. Bellare, P. Rogaway, The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg, Germany, San Francisco, CA, USA (Apr 8–12, 2001)

[2] N. Aviram, K. Gellert, T. Jager, Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 117–150. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)

[3] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J.A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, Y. Shavitt, DROWN: Breaking TLS using SSLv2. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 689–706. USENIX Association, Austin, TX, USA (Aug 10–12, 2016)

[4] C. Bader, D. Hofheinz, T. Jager, E. Kiltz, Y. Li, Tightly-secure authenticated key exchange. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 629–658. Springer, Heidelberg, Germany, Warsaw, Poland (Mar 23–25, 2015)

[5] C. Bader, T. Jager, Y. Li, S. Schäge, On the impossibility of tight cryptographic reductions. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 273–304. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016)

[6] M. Bellare, New proofs for NMAC and HMAC: Security without collision resistance. *J. Cryptol.* **28**(4), 844–878 (2015)

[7] M. Bellare, R. Canetti, H. Krawczyk, Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996)

[8] M. Bellare, R. Canetti, H. Krawczyk, Pseudorandom functions revisited: The cascade construction and its concrete security. In: 37th FOCS. pp. 514–523. IEEE Computer Society Press, Burlington, Vermont (Oct 14–16, 1996b)

[9] M. Bellare, T. Ristenpart, Simulation without the artificial abort: Simplified proof and improved concrete security for waters' IBE scheme. Cryptology ePrint Archive, Report 2009/084 (2009) http://eprint.iacr.org/2009/084

[10] M. Bellare, T. Ristenpart, Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 407–424. Springer, Heidelberg, Germany, Cologne, Germany (Apr 26–30, 2009)

[11] M. Bellare, P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press, Fairfax, Virginia, USA (Nov 3–5, 1993)

[12] M. Bellare, P. Rogaway, Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994)

[13] M. Bellare, B. Tackmann, The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 247–276. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016)

[14] D.J. Bernstein, N. Duif, T. Lange, P. Schwabe, B.Y. Yang, High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg, Germany, Nara, Japan (Sep 28–Oct 1, 2011)

[15] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.Y. Strub, J.K. Zinzindohoue, A messy state of the union: Taming the composite state machines of TLS. In: 2015 IEEE Symposium on Security and Privacy. pp. 535–552. IEEE Computer Society Press, San Jose, CA, USA (May 17–21, 2015)

[16] K. Bhargavan, C. Fournet, M. Kohlweiss, miTLS: Verifying protocol implementations against real-world attacks. *IEEE Secur. Privacy* **14**(6), 18–25 (2016)

[17] K. Bhargavan, C. Brzuska, C. Fournet, M. Green, M. Kohlweiss, S. Zanella-Béguelin, Downgrade resilience in key-exchange protocols. In: 2016 IEEE Symposium on Security and Privacy. pp. 506–525. IEEE Computer Society Press, San Jose, CA, USA (May 22–26, 2016)

[18] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, P.Y. Strub, Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In: 2014 IEEE Symposium on Security and Privacy. pp. 98–113. IEEE Computer Society Press, Berkeley, CA, USA (May 18–21, 2014)

[19] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.Y. Strub, Zanella Béguelin, S.: Proving the TLS handshake secure (as it is). In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 235–255. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014)

[20] D. Boneh, The decision Diffie-Hellman problem. In: Third Algorithmic Number Theory Symposium (ANTS). LNCS, vol. 1423. Springer, Heidelberg, Germany (1998), invited paper

[21] J. Brendel, M. Fischlin, F. Günther, C. Janson, PRF-ODH: Relations, instantiations, and impossibility results. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 651–681. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)

[22] C. Brzuska, M. Fischlin, N. Smart, B. Warinschi, S. Williams, Less is more: Relaxed yet composable security notions for key exchange. Cryptology ePrint Archive, Report 2012/242 (2012), http://eprint.iacr.org/2012/242

[23] C. Brzuska, M. Fischlin, B. Warinschi, S.C. Williams, Composability of Bellare-Rogaway key exchange protocols. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011. pp. 51–62. ACM Press, Chicago, Illinois, USA (Oct 17–21, 2011)

[24] S. Chen, S. Jero, M. Jagielski, A. Boldyreva, C. Nita-Rotaru, Secure communication channel establishment: TLS 1.3 (over TCP fast open) vs. QUIC. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019, Part I. LNCS, vol. 11735, pp. 404–426. Springer, Heidelberg, Germany, Luxembourg (Sep 23–27, 2019)

[25] K. Cohn-Gordon, C. Cremers, K. Gjøsteen, H. Jacobsen, T. Jager, Highly efficient key exchange protocols with optimal tightness. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 767–797. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019)

[26] J.S. Coron, Optimal security proofs for PSS and other signature schemes. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 272–287. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Apr 28–May 2, 2002)

[27] H. Davis, F. Günther, Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029 (2020), https://eprint.iacr.org/2020/1029

[28] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Zanella-Béguelin, K. Bhargavan, J. Pan, J.K. Zinzindohoue, Implementing and proving the TLS 1.3 record layer. In: 2017 IEEE Symposium on Security and Privacy. pp. 463–482. IEEE Computer Society Press, San Jose, CA, USA (May 22–26, 2017)

[29] W. Diffie, M.E. Hellman, New directions in cryptography. *IEEE Trans. Inf. Theory* **22**(6), 644–654 (1976)

[30] Y. Dodis, J.P. Steinberger, Message authentication codes from unpredictable block ciphers. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 267–285. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009)

[31] B. Dowling, M. Fischlin, F. Günther, D. Stebila, A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1197–1210. ACM Press, Denver, CO, USA (Oct 12–16, 2015)

[32] B. Dowling, M. Fischlin, F. Günther, D. Stebila, A cryptographic analysis of the TLS 1.3 handshake protocol candidates. Cryptology ePrint Archive, Report 2015/914 (2015) http://eprint.iacr.org/2015/914

[33] B. Dowling, M. Fischlin, F. Günther, D. Stebila, A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081 (2016), http://eprint.iacr.org/2016/081

[34] B. Dowling, D. Stebila, Modelling ciphersuite and version negotiation in the TLS protocol. In: Foo, E., Stebila, D. (eds.) ACISP 15. LNCS, vol. 9144, pp. 270–288. Springer, Heidelberg, Germany, Brisbane, QLD, Australia (Jun 29–Jul 1, 2015)

[35] M. Fersch, E. Kiltz, B. Poettering, On the one-per-message unforgeability of (EC)DSA and its variants. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part II. LNCS, vol. 10678, pp. 519–534. Springer, Heidelberg, Germany, Baltimore, MD, USA (Nov 12–15, 2017)

[36] M. Fischlin, F. Günther, Multi-stage key exchange and the case of Google's QUIC protocol. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 2014. pp. 1193–1204. ACM Press, Scottsdale, AZ, USA (Nov 3–7, 2014)

[37] M. Fischlin, F. Günther, Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In: IEEE European Symposium on Security and Privacy 2017, EuroS&P 2017, Paris, France, April 26–28, 2017. pp. 60–75. IEEE (2017)

[38] K. Gjøsteen, T. Jager, Practical and tightly-secure digital signatures and authenticated key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 95–125. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)

[39] S. Goldwasser, S. Micali, R.L. Rivest, A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* **17**(2), 281–308 (1988)

[40] F. Günther, Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols. Ph.D. thesis, Darmstadt University of Technology, Germany (2018), http://tuprints.ulb.tu-darmstadt.de/7162/

[41] V.T. Hoang, S. Tessaro, A. Thiruvengadam, The multi-user security of GCM, revisited: Tight bounds for nonce randomization. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1429–1440. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018)

[42] T. Jager, S.A. Kakvi, A. May, On the security of the PKCS#1 v1.5 signature scheme. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1195–1208. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018)

[43] T. Jager, F. Kohlar, S. Schäge, J. Schwenk, On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)

[44] T. Jager, J. Schwenk, J. Somorovsky, On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1185–1196. ACM Press, Denver, CO, USA (Oct 12–16, 2015)

[45] D. Johnson, A. Menezes, S.A. Vanstone, The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Secur.* **1**(1), 36–63 (2001), https://doi.org/10.1007/s102070100002

[46] S. Josefsson, I. Liusvaara, Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032 (Informational) (Jan 2017), https://www.rfc-editor.org/rfc/rfc8032.txt

[47] S.A. Kakvi, On the security of RSA-PSS in the wild. In: Proceedings of the 5th Security Standardisation Research Workshop (SSR–19), November 11, 2019, London, United Kingdom. (2019)

[48] B. Kaliski, PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational) (Mar 1998), https://www.rfc-editor.org/rfc/rfc2313.txt, obsoleted by RFC 2437

[49] H. Krawczyk, M. Bellare, R. Canetti: HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational) (Feb 1997), https://www.rfc-editor.org/rfc/rfc2104.txt, updated by RFC 6151

[50] H. Krawczyk, P. Eronen, HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational) (May 2010), https://www.rfc-editor.org/rfc/rfc5869.txt

[51] H. Krawczyk, SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 400–425. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003)
[52] H. Krawczyk, Cryptographic extraction and key derivation: The HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 2010)
[53] H. Krawczyk, Cryptographic extraction and key derivation: The HKDF scheme. Cryptology ePrint Archive, Report 2010/264 (2010b), http://eprint.iacr.org/2010/264
[54] H. Krawczyk, A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in TLS 1.3). In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1438–1450. ACM Press, Vienna, Austria (Oct 24–28, 2016)
[55] H. Krawczyk, K.G. Paterson, H. Wee, On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 429–448. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013)
[56] H. Krawczyk, H. Wee, The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978 (2015), http://eprint.iacr.org/2015/978
[57] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, B. Preneel, A cross-protocol attack on the TLS protocol. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012. pp. 62–72. ACM Press, Raleigh, NC, USA (Oct 16–18, 2012)
[58] D. Micciancio, M. Walter, On the bit security of cryptographic primitives. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 3–28. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29–May 3, 2018)
[59] K. Moriarty (Ed.), B. Kaliski, J. Jonsson, A. Rusch, PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017 (Informational) (Nov 2016), https://www.rfc-editor.org/rfc/rfc8017.txt
[60] P. Morrissey, N.P. Smart, B. Warinschi, A modular security analysis of the TLS handshake protocol. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 55–73. Springer, Heidelberg, Germany, Melbourne, Australia (Dec 7–11, 2008)
[61] N. Nisan, A. Ta-Shma, Extracting randomness: A survey and new constructions. *J. Comput. Syst. Sci.* **58**(1), 148–173 (1999), https://doi.org/10.1006/jcss.1997.1546
[62] N. Nisan, D. Zuckerman, Randomness is linear in space. *J. Comput. Syst. Sci.* **52**(1), 43–52 (1996), https://doi.org/10.1006/jcss.1996.0004
[63] T. Okamoto, D. Pointcheval, The gap-problems: A new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg, Germany, Cheju Island, South Korea (Feb 13–15, 2001)
[64] E. Rescorla, Keying Material Exporters for Transport Layer Security (TLS). RFC 5705 (Proposed Standard) (Mar 2010), https://www.rfc-editor.org/rfc/rfc5705.txt, updated by RFCs 8446, 8447
[65] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard) (Aug 2018), https://www.rfc-editor.org/rfc/rfc8446.txt
[66] P. Rogaway, Formalizing human ignorance. In: Nguyen, P.Q. (ed.) Progress in Cryptology - VIETCRYPT 06. LNCS, vol. 4341, pp. 211–228. Springer, Heidelberg, Germany, Hanoi, Vietnam (Sep 25–28, 2006)
[67] P. Rogaway, Formalizing human ignorance: Collision-resistant hashing without the keys. Cryptology ePrint Archive, Report 2006/281 (2006b), http://eprint.iacr.org/2006/281
[68] V. Shoup, Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), http://eprint.iacr.org/2004/332
[69] D. Wagner, B. Schneier, Analysis of the SSL 3.0 protocol. In: Proceedings of the 2nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2. pp. 29–40. WOEC–96, USENIX Association, USA (November 1996)