A preliminary version of this paper appears in the proceedings of the *22nd ACM Conference on Computer and Communications Security (CCS 2015)*, DOI: 10.1145/2810103.2813653. This is the full version.

A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates

Benjamin Dowling[1] Marc Fischlin[2] Felix Günther[2] Douglas Stebila[1]

[1] Queensland University of Technology, Brisbane, Australia
[2] Cryptoplexity, Technische Universität Darmstadt, Germany
b1.dowling@qut.edu.au, marc.fischlin@cryptoplexity.de,
guenther@cs.tu-darmstadt.de, stebila@qut.edu.au

February 1, 2016

Abstract. The Internet Engineering Task Force (IETF) is currently developing the next version of the Transport Layer Security (TLS) protocol, version 1.3. The transparency of this standardization process allows comprehensive cryptographic analysis of the protocols prior to adoption, whereas previous TLS versions have been scrutinized in the cryptographic literature only after standardization. Here we look at two related, yet slightly different candidates which were in discussion for TLS 1.3 at the point of writing of the main part of the paper in May 2015, called `draft-ietf-tls-tls13-05` and `draft-ietf-tls-tls13-dh-based`.

We give a cryptographic analysis of the primary ephemeral Diffie–Hellman-based handshake protocol, which authenticates parties and establishes encryption keys, of both TLS 1.3 candidates. We show that both candidate handshakes achieve the main goal of providing secure authenticated key exchange according to an augmented multi-stage version of the Bellare–Rogaway model. Such a multi-stage approach is convenient for analyzing the design of the candidates, as they establish multiple session keys during the exchange.

An important step in our analysis is to consider compositional security guarantees. We show that, since our multi-stage key exchange security notion is composable with arbitrary symmetric-key protocols, the use of session keys in the record layer protocol is safe. Moreover, since we can view the abbreviated TLS resumption procedure also as a symmetric-key protocol, our compositional analysis allows us to directly conclude security of the combined handshake with session resumption.

We include a discussion on several design characteristics of the TLS 1.3 drafts based on the observations in our analysis.

Keywords. Transport Layer Security (TLS), key exchange, protocol analysis, composition

Contents

1 Introduction

The *Transport Layer Security (TLS)* protocol is one of the most widely deployed cryptographic protocols in practice, protecting numerous web and e-mail accesses every day. The TLS *handshake protocol* allows a client and a server to authenticate each other and to establish a key, and the subsequent *record layer protocol* provides confidentiality and integrity for communication of application data. Despite its large-scale deployment, or perhaps because of it, we have witnessed frequent successful attacks against TLS. In the past few years alone, there have been many practical attacks that have received significant attention, either exploiting weaknesses in underlying cryptographic primitives (such as weaknesses in RC4 [ABP+13]), errors in the design of the TLS protocol (BEAST [Duo11], the Lucky 13 attack [AP13], the triple handshake attack [BDF+14], the POODLE attack [MDK14], the Logjam attack [ABD+15]), or flaws in implementations (the Heartbleed attack [Cod14], state machine attacks (SMACK [BBDL+15])). Some of these attacks apply only to earlier versions of the TLS protocol, but for legacy reasons many parties still support versions older than the latest one, TLS 1.2.

1.1 Towards the New Standard TLS 1.3

Partly due to the above security problems with the existing versions of TLS, but also because of additional desirable privacy features and functional properties such as low handshake latency, the Internet Engineering Task Force (IETF) is currently drafting a new TLS 1.3 standard. As of May 2015, there were two (slightly different) candidates in discussion: one is `draft-ietf-tls-tls13-05` [Res15a] (which we shorten to `draft-05`), the other one is the forked `draft-ietf-tls-tls13-dh-based` [Res15c] (which we shorten to `draft-dh`), incorporating a different key schedule based on ideas by Krawczyk and Wee.[1] In this work, we provide a comprehensive cryptographic evaluation of the primary Diffie–Hellman-based handshake of both drafts.[2] We believe that it is important that cryptographic evaluation take place *before* standardization. This contrasts with the history of TLS and its predecessor the Secure Sockets Layer (SSL) protocol: SSL 3 was standardized in 1996, TLS 1.0 in 1999, TLS 1.1 in 2006, and TLS 1.2 in 2008, but the first comprehensive cryptographic proof of any complete TLS ciphersuite did not appear until 2012 [JKSS12].

The protocol design in both TLS 1.3 drafts includes several cryptographic changes that are substantially different from TLS 1.2, including: (1) encrypting some handshake messages with an intermediate session key, to provide confidentiality of handshake data such as the client certificate; (2) signing the entire handshake transcript for authentication; (3) including hashes of handshake messages in a variety of key calculations; (4) encrypting the final `Finished` messages in the handshake with a different key than is used for encrypting application data; (5) deprecating a variety of cryptographic algorithms (including RSA key transport, finite-field Diffie–Hellman key exchange, SHA-1, RC4, CBC mode, MAC-then-encode-then-encrypt); (6) using modern authenticated encryption with associated data (AEAD) schemes for symmetric encryption; and (7) providing handshakes with fewer message flows to reduce latency.

These changes are meant in part to address several of the aforementioned attacks. While some of those attacks are implementation-specific and escape abstract cryptographic evaluation, assessing the cryptographic security of the design of TLS 1.3[3] can provide assurance that the protocol design does not display any unexpected cryptographic weaknesses. Our goal is a comprehensive assessment of the security of the handshake protocol in `draft-05` and `draft-dh`. We focus solely on the handshake protocol as a key exchange protocol; these drafts provide a cleaner separation between the key exchange in the

[1] Since May 2015, several follow-up draft versions of TLS 1.3 have been published that build on `draft-05` but incorporate major changes including the `draft-dh` key schedule. We recently adapted our analysis [DFGS16] to cover the design of `draft-10` [Res15b].

[2] `draft-05` foresees and `draft-dh` sketches a subordinate pre-shared key variant relying on previously shared keys. Later drafts merge pre-shared key and session resumption.

[3] When we refer to "TLS 1.3", we mean the common features of `draft-05` and `draft-dh`.

handshake protocol and the use of the resulting session key in the record layer protocol. This contrasts with TLS 1.2 and earlier, where the session key was used both for record layer encryption and encryption of the `Finished` messages in the handshake, making it impossible for TLS 1.2 to satisfy standard key exchange indistinguishability notions and requiring either (a) a more complex security model that treats the handshake and record layer together [JKSS12] or (b) a cunning approach to release the record layer key early [BFK+14] (see also Section 1.4). The cleaner separation in the TLS 1.3 design allows us to take a *compositional approach* to the security of TLS 1.3, treating the handshake separate from the record layer, and also allowing us to include session resumption for abbreviated handshakes.

1.2 Modeling TLS 1.3 as a Multi-Stage Key Exchange Protocol

The message flow for both drafts is similar and shown in Figures 1 and 2 along with the respective key schedule. It is convenient to view TLS 1.3 as a *multi-stage* key exchange protocol [FG14] in which both parties, the client and the server, agree on multiple session keys, possibly using one key to derive the next one. In the first stage, the first session key is derived via an anonymous Diffie–Hellman key exchange (in the `ClientKeyShare` and `ServerKeyShare` messages) from which a handshake master secret HMS is computed. This handshake master secret is used to compute a handshake traffic key tk_{hs} which encrypts the remaining messages of the handshake and should provide some form of outsider privacy for the exchanged certificates.

In the second stage, the parties (depending on the desired authentication level) exchange signatures over the (hash of the) transcript under a certified key in order to authenticate. They then derive the application traffic key tk_{app} for securing the application messages, the resumption master secret RMS if they resume sessions, and the exporter master secret EMS which can be used for deriving additional keying material. Viewing each of the keys as one of the multi-stage session keys enables us to argue about their security, even if the other keys are leaked. Both parties conclude the protocol by exchanging `Finished` messages over the transcripts, generated using HMS or a separate key.

1.3 Our Results

Security of draft-05 and draft-dh full handshakes. First, we show (in Sections 5 and 6) that both TLS 1.3 drafts are secure multi-stage key exchange protocols where different stages and simultaneous runs of the protocols can be unauthenticated, unilaterally authenticated, or mutually authenticated. On a high level, this means that the handshakes establish record layer keys, resumption keys, and exporter keys that look random to an adversary. This holds even with sessions that run concurrently and if the adversary controls the whole network, is able to corrupt the long-term secret keys of other parties, and allowed to reveal keys established in other sessions, thus providing quite strong security guarantees for practice. Moreover, the multi-stage model used allows us to show that even leakage of record layer or exporter keys in the same handshake session do not compromise each other's security.

This requires some additions to the multi-stage key exchange security model of Fischlin and Günther [FG14] to allow for unauthenticated sessions and post-specified peers, as described in Section 4, as well as to handle authentication based on pre-shared symmetric keys, as described in Section 8. Notably, our security proof only relies on so-called standard cryptographic assumptions such as the Decisional Diffie–Hellman (DDH) assumption, unforgeability of the deployed signature scheme, collision resistance of the hash function, and pseudorandomness of the key derivation function. This is in sharp contrast to many other key exchange protocols, where often the key derivation function is modeled as a random oracle. The cryptographic analysis of signed-Diffie–Hellman ciphersuites in TLS 1.2 required an uncommon (yet not implausible) pseudorandom-oracle Diffie–Hellman assumption [JKSS12, KPW13].

Composition theorem for use of session keys. In order to show that the keys established in TLS 1.3's multi-stage key exchange handshake can be safely used in the record layer encryption, we extend the composition frameworks of Brzuska et al. [BFWW11] and Fischlin and Günther [FG14] in Section 7 to multi-stage key exchange protocols with mixed unauthenticated, unilateral, and mutual authentication.

A key point to secure composition of multi-stage key agreement protocols with arbitrary symmetric-key protocols in [FG14] is (session-)key independence. This roughly means that one can reveal a session key without endangering the security of future session keys. Both TLS 1.3 drafts satisfy this, enabling us to argue about secure composition of the full handshake protocols with, say, a secure channel protocol.[4]

Recent work by Badertscher et al. [BMM+15] shows that the authenticated encryption (with associated data) used in the record layer in both TLS 1.3 drafts is secure. See also Fischlin et al. [FGMP15] for reassuring results about the design of the record layer protocol when viewed in terms of data streams. Our compositional approach immediately implies that the application traffic keys output by both drafts' handshakes can be safely used in the record layer.

Security of session resumption in TLS 1.3 drafts. TLS includes a mode for *abbreviated handshakes*, in which parties who have previously established a session can save round trips and computation by using the previous key as the basis for a new session; this is called *session resumption*. We can treat the abbreviated handshake as a separate symmetric-key protocol (modeled in Section 8) with an independent, modular security analysis (in Section 9), then use our compositional approach to show that the resumption master secrets output by the full handshake can be safely composed with the abbreviated handshake.

Comments on the design of TLS 1.3. Our results allow us to give insight on some of the design choices of TLS 1.3, such as the role of the `Finished` messages and of the new session hash. Those comments follow in Section 3, immediately after we review the structure of the TLS 1.3 handshakes in the next section.

1.4 Related Work

A significant step forward to a comprehensive analysis of TLS 1.2 handshake protocol and its implementation came with the recent work of Bhargavan et al. [BFK+14], who analyze the TLS 1.2 handshake protocol in the agile setting, covering the various ciphersuite options in TLS 1.2, and applying the results to a miTLS implementation [FKS11, BFK+13]. Using epochs and shared states between executions they also capture resumption and renegotiation in TLS 1.2. We are not aware of extensions of their result to the TLS 1.3 candidates.

A key point in the cryptographic analysis of Bhargavan et al. [BFK+14] is to overcome the issue that the session key is already used in the final part of the handshake protocol in TLS 1.2, by separating these steps from the rest of the handshake. They achieve this by using a different notion than session identifiers to safely determine partners, called (peer-)exchange variables. While the designers for TLS 1.3 have eliminated the `Finished`-message problem, avoiding the usage of the session key in the handshake entirely, the approach of switching to alternative notions of safe partnering turns out to be useful for our setting, too. We introduce the notion of contributive identifiers which can be roughly thought of as "partial" session identifiers. In contrast to [BFK+14] our contributive identifiers resemble much closer the common session identifiers (which we also keep in our model to define session partners), and we only use contributive identifiers for non–mutually authenticated sessions and our compositional result.

[4] A technical nuisance in this regard is that, for the key independence property to hold, we cannot use the common approach to define session partnering via the (here partly encrypted) communication transcript, but need to base session partnering on the unencrypted key exchange data. This exacerbates for example the application of the hybrid method to go from multiple tested session keys to a single test queries.

A main difference to the epoch-based analysis in [BFK+14] is that we use a general composition result to deal with resumption. That is, we view the resumption step in TLS 1.3 as a special symmetric-key protocol which follows the handshake. The approach in [BFK+14] is to consider resumption as an abbreviated handshake protocol variant, executed in a different epoch. Finally, let us remark that the analysis in [BFK+14] extends to the implementation level, whereas our results are purely on the abstract level.

Concurrently to our work, Kohlweiss et al. [KMO+14] transferred their constructive-cryptography based analysis of TLS 1.2 to (a modified version of) the `draft-05` version of TLS 1.3, where they assume that the second-stage messages are actually sent unencrypted. They do not consider the `draft-dh` draft, nor do they cover the resumption step. However, our approach of integrating resumption via composition may also be viable for their model.

1.5 Limitations

Since TLS 1.3 is still a work in progress, our analysis is inevitably limited to the draft specifications available at the time of writing, and the actual TLS 1.3 may eventually differ from the draft versions we have analyzed. Nonetheless, this paper's analysis can provide insight into the design of the existing drafts. We believe it is imperative for the cryptographic community to be engaged in the design and analysis of TLS 1.3 before, rather than after, it is standardized.

One of the aspired design goals of TLS 1.3 is to support the possibility for zero round-trip time (0-RTT) for the handshake protocol, which would enable transmission of application from the client to the server on the first message flow, saving latency. This unfortunately comes with inherent problems, namely, lack of forward secrecy and the possibility of replay attacks. `draft-05` provides no specification for 0-RTT handshakes; `draft-dh` introduces an extra "semi-static" public key for this purpose, however at the time of writing `draft-dh` does not provide sufficient protocol detail to allow a full cryptographic analysis of this. We do not model leakage of the `draft-dh` semi-static key at this point as it plays no role (for secrecy) in our security analysis, but defer carefully crafting reasonable conditions for its exposure until the 0-RTT handshake is specified completely.

As noted above, our compositional approach allows for the separate analysis of the full handshake, the record layer, and the session resumption handshake, and then composition shows that the various keys output from the handshake can be safely used with the record layer encryption and session resumption. This suggests the following approach to prove the full TLS protocol suite to be secure: show that session resumption itself constitutes a secure key exchange protocol (with a pre-shared symmetric key which comes from the handshake protocol here), compose it securely with the record layer protocol, and then "cascade" this composed symmetric-key protocol with the compositional handshake protocol. Unfortunately, one limitation of the current composition frameworks is that composition is only supported between a key exchange protocol *with forward secrecy* and an arbitrary symmetric key protocol. This holds here for the main handshake protocol and allows us to immediately argue secure composition with session resumption or with the record layer. However, session resumption does not provide forward secrecy (with respect to corruption of the resumption (pre-)master secrets), so we cannot automatically conclude safe use of the session keys output by session resumption in the record layer. Extending the composition framework to support multi-stage key exchange protocols without forward secrecy is left for future work.

2 The TLS 1.3 Handshake Protocol

For both `draft-05` and `draft-dh`, the handshake protocol is divided into two phases: the *negotiation* phase, where parties negotiate ciphersuites and key-exchange parameters, generate unauthenticated shared key material, and establish handshake traffic keys; and the *authentication* phase, where parties authenticate

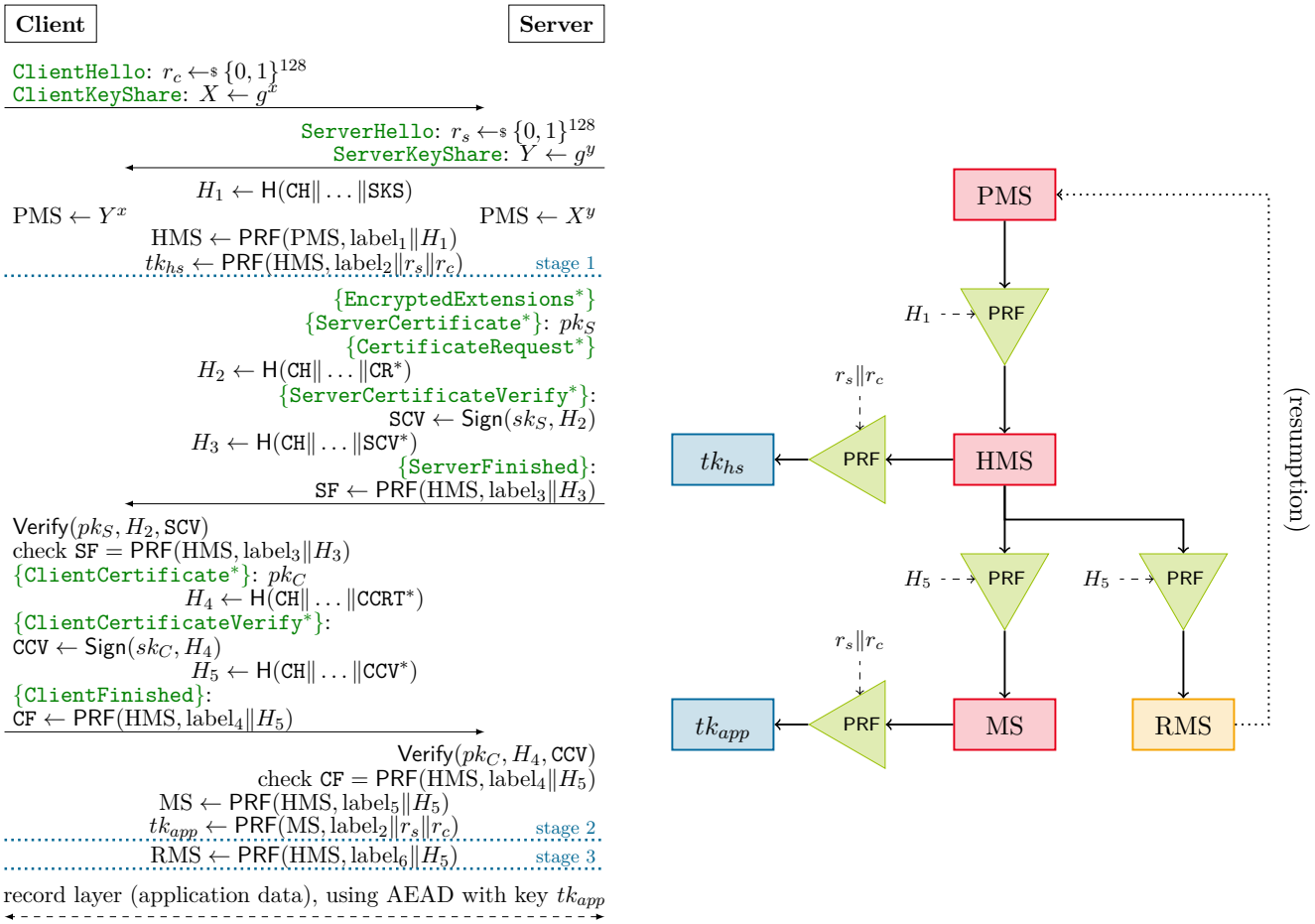

Figure 1: The handshake protocol in TLS 1.3 `draft-05` (left) and its key schedule (right). `XXX`: Y denotes TLS message `XXX` containing Y. {XXX} indicates a message `XXX` encrypted using AEAD encryption with handshake traffic key tk_{hs}. `XXX`* indicates a message that is only sent in unilateral or mutual authentication modes. In the key schedule, dotted-line input to PRF is the input value (omitting the label as additional input).

the handshake transcript according to the authentication properties negotiated earlier and output authenticated application traffic keys, independent from the previous handshake traffic keys.

2.1 draft-05 Handshake

Figure 1 shows the message flow and relevant cryptographic computations as well as the key schedule for the full handshake in `draft-05`.

The handshake messages are as follows:

- `ClientHello` (`CH`)/`ServerHello` (`SH`) contain the supported versions and ciphersuites for negotiation purposes, as well as random nonces r_c resp. r_s. `SH` can contain a session identifier `session_id` field for future session resumption. Both `CH` and `SH` can also include various extension fields.
- `ClientKeyShare` (`CKS`)/`ServerKeyShare` (`SKS`) contain the ephemeral Diffie–Hellman shares $X = g^x$ resp. $Y = g^y$ for one or more groups selected by an extension in `CH`/`SH`.

Both parties can now compute the premaster secret PMS as the Diffie–Hellman shared secretg^{xy} and then use a pseudorandom function PRF to compute a handshake master secret HMS and handshake traffic key tk_{hs}; both are unauthenticated at this point.

All subsequent messages are encrypted using tk_{hs}:

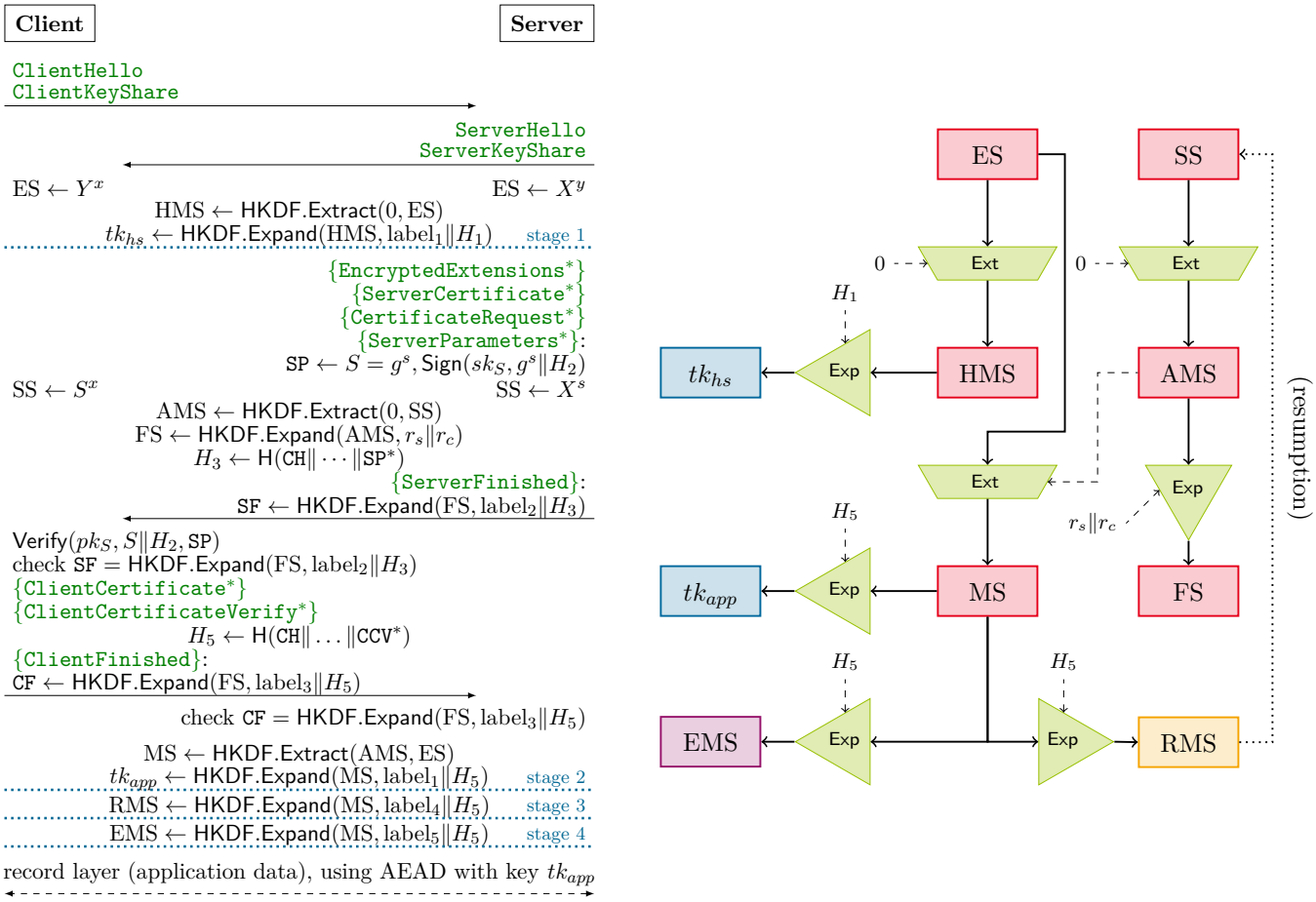

Figure 2: The handshake protocol in TLS 1.3 `draft-dh` (left) and its key schedule (right). Hash value and message computations not stated explicitly are performed identically to TLS 1.3 `draft-05` (Figure 1). For unauthenticated handshakes, the `ServerCertificate` and `ServerParameters` messages are omitted and key derivation proceeds with SS set to ES. In the key schedule, Ext and Exp are short for HKDF.Extract resp. HKDF.Expand. Dotted-line input to Ext is the (extractor) salt, dotted-line input to Exp is the (context) information input; label inputs are omitted.

- `EncryptedExtensions` (`EE`) contains more extensions.
- `ServerCertificate` (`SCRT`)/`ClientCertificate` (`CCRT`) contain the public-key certificate of the respective party.
- `CertificateRequest` (`CR`) indicates the server requests that the client authenticates using a certificate.
- `ServerCertificateVerify` (`SCV`)/`ClientCertificateVerify` (`CCV`) contain a digital signature over the *session hash* (the hash of all handshakes messages sent and received at that point in the protocol run).
- `ClientFinished` (`CF`)/`ServerFinished` (`SF`) contain the PRF evaluation on the session hash keyed with HMS.

Both parties can now compute the master secret MS and the application traffic key tk_{app} as well as the resumption master secret RMS for use in future session resumptions.

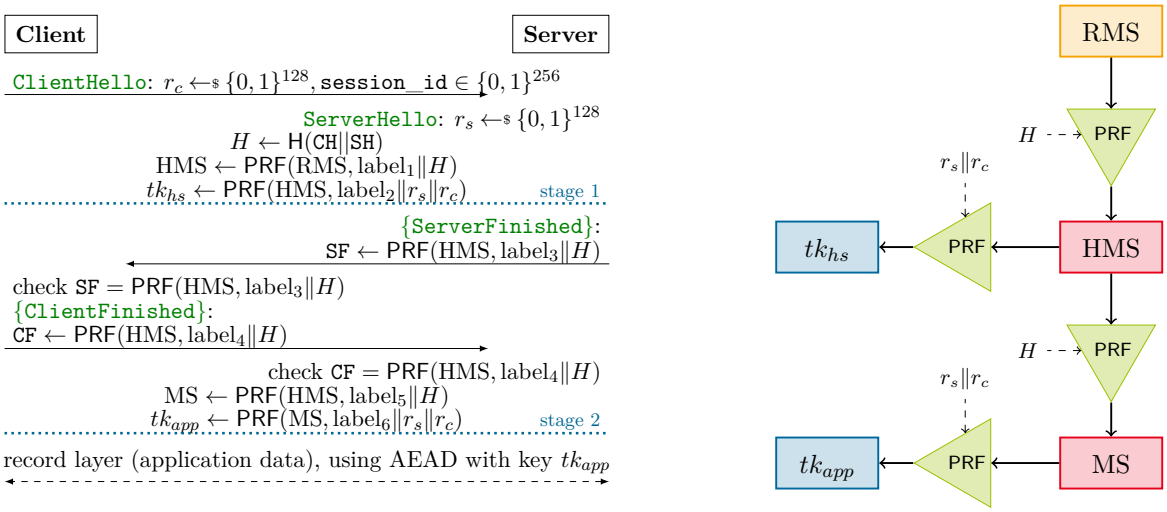

Figure 3: Session resumption in TLS 1.3 `draft-05` using the resumption master secret RMS as a pre-shared key (left) and its key schedule (right). In the key schedule, dotted-line input to PRF is the input value (omitting the label as additional input).

2.2 draft-dh Handshake

Figure 2 shows the message flow and cryptographic computations as well as the key schedule for the full handshake in `draft-dh`. The main difference to `draft-05` is the `ServerParameters` message (replacing `SCV`) containing the server's additional semi-static Diffie–Hellman share, allowing the application traffic keys to rely on both ephemeral and non-ephemeral secrets. Moreover, key derivation is done using the HKDF extract-then-expand key derivation function [Kra10], rather than the TLS PRF.

We adopt here the standard notation for the two HKDF functions: $\mathsf{HKDF.Extract}(XTS, SKM)$ on input an (non-secret and potentially fixed) extractor salt XTS and some source key material SKM outputs a pseudorandom key PRK. $\mathsf{HKDF.Expand}(PRK, CTXinfo)$ on input a pseudorandom key PRK (from the Extract step) and some (potentially empty) context information $CTXinfo$ outputs key material KM.[5]

2.3 Session Resumption

Session resumption in `draft-05` has similarly been changed from TLS 1.2 to separate handshake and application traffic keys. As shown in Figure 3, `ClientHello` includes a preshared-secret identifier (referred to in the drafts as the "session identifier `session_id`") of some previously established session. The client and server use that previous session's resumption master secret, which has previously been authenticated, so they do not exchange key shares or signatures, and fresh nonces r_c, r_s to derive the new keys.

The differences between `draft-05` session resumption and `draft-dh` session resumption are limited to secret and key computation as well as `CF` and `SF` computation (again, $H = \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$):

1. $\mathrm{AMS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{RMS})$
2. $tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{AMS}, \mathrm{label}_1\|H)$
3. $\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{AMS}, \mathrm{label}_2\|r_s\|r_c)$
4. $\mathtt{SF} \leftarrow \mathsf{HKDF.Expand}(\mathrm{FS}, \mathrm{label}_3\|H)$
5. $\mathtt{CF} \leftarrow \mathsf{HKDF.Expand}(\mathrm{FS}, \mathrm{label}_4\|H)$
6. $\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{AMS})$
7. $tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_5\|H)$
8. $\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_6\|H)$

[5] For simplicity, we omit the original third parameter L in Expand determining its output length and always assume that $L = \lambda$ for our security parameter λ.

3 Comments on the TLS 1.3 Design

Our analysis provides several insights into the TLS 1.3 drafts, for both the basic cryptographic choices, as well as for the yet to be fully specified 0-RTT versions.

3.1 Basic Handshake Protocols

Soundness of key separation. Earlier versions of TLS used in the same session key to encrypt the application data as well as the `Finished` messages at the end of the handshake. This made it impossible to show that the TLS session key satisfied standard Bellare–Rogaway-style key indistinguishability security [BR94], and necessitated non-standard assumptions in the security proof (e.g., the PRF-Oracle-Diffie–Hellman (PDF-ODH) assumption [JKSS12, KPW13]). We confirm that the change in keys for encryption of handshake messages allows both TLS 1.3 drafts to achieve standard key indistinguishability security without non-standard assumptions.

Key independence. Both TLS 1.3 drafts achieve key independence in the multi-stage security setting, which heavily strengthens their overall security. (Recall key independence is the property that one can reveal a session key without endangering the security of later-stage keys.) Beyond making it amenable to generic composition, key independence safeguards the usage of derived keys against inter-protocol effects of security breakdowns.

`draft-dh` takes a slightly more composable approach to keying material exporting than `draft-05`. In `draft-dh`, an exporter master secret EMS is derived from the master secret and then applications get exported keying material as $\mathsf{PRF}(\mathrm{EMS}, \mathrm{label})$. In `draft-05`, applications get exported keying material directly as $\mathsf{PRF}(\mathrm{MS}, \mathrm{label})$. Key independence in the `draft-dh` approach allows us to treat derivation of exported keying material as a separate symmetric protocol, whereas in `draft-05` each exported key must be considered in the main analysis, so we argue the `draft-dh` approach of a separate exporter master secret is preferable.

Signing the session hash. In both TLS 1.3 drafts, authenticating parties (the server, and sometimes the client) sign (the hash of) all handshake messages up to when the signature is issued. This is different from TLS 1.2 and earlier, where the server's signature is only over the client and server random nonces and the server's ephemeral public key. The server signing the transcript makes our proof easier and allows us to rely on standard cryptographic assumptions.

Encryption of handshake messages. Both TLS 1.3 drafts encrypt the second part of the handshake using the initial handshake traffic key tk_{hs}, aiming to provide some form of privacy (against passive adversaries) for these messages, in particular for the server and client certificates. Our analysis shows that the handshake traffic key does indeed have security against passive adversaries and hence increases the handshake's privacy. The secrecy of the final session keys however does not rely on the handshake being encrypted and would remain secure even if was done in clear. Our analysis considers the encrypted case, showing that this encryption does not negatively affect the security goals.

Finished messages. The `Finished` messages in both drafts are computed by applying the PRF (or HKDF in `draft-dh`) to the (hash of the) handshake transcript. Interestingly, according to our proof the `Finished` messages do not contribute to the session key secrecy in the full handshake or the session resumption handshake in the sense that the key exchange would be secure without these messages. This is mainly because the signatures already authenticate the transcripts. This contrasts with the case of RSA key transport in the TLS 1.2 full handshake: the analyses of both Krawczyk et al. [KPW13] and Bhargavan

et al. [BFK+14] note potential weaknesses or require stronger security assumptions if `Finished` messages are omitted. From an engineering perspective, the `Finished` messages can still be interpreted as providing some form of (explicit) session key confirmation, but is not cryptographically required to achieve key indistinguishability. In session resumption, the `Finished` messages give the only explicit authentication.

Session hash in key derivation. Both TLS 1.3 drafts include a hash of all messages exchanged so far in the derivation of all session keys and, in `draft-05`, also in deriving the master secrets. This session hash was introduced in response to the triple handshake attack [BDF+14] on TLS 1.2 and earlier, with the goal of ensuring that sessions with different session identifiers have different master secrets. In our security analysis of both full handshakes, the online signatures computed over the handshake messages already suffice to bind the exchanged messages to the authenticated parties and established keys, so including the session hash in the key derivations does not contribute to the session keys' secrecy. If keys are meant to be used as a channel identifier or for channel binding (with the purpose of leveraging the sesssion protection and authentication properties established by TLS in an application-layer protocol), including the session hash is appropriate. While the standardized `tls-unique` [AWZ10] and proposed `tls-unique-prf` [Jos15] TLS channel binding methods do not use keys directly for binding, the low cost of including the session hash seems worth it in case an application developer decides to use keying material directly for binding. Interestingly, in the `draft-dh` version, computing the master secret requires a semi-static secret and ephemerally-generated secret, but does not include the session hash. In `draft-dh` session resumption, there is no ephemeral shared secret and the master secret is computed as a series of HKDF.Extract computations over a 0-string using the resumption secret as the key. All sessions sharing the same resumption master secret then compute the same master secret. However, since key derivation still uses the session hash as context, keys are unique assuming uniqueness of protocol messages (assured, e.g., via unique nonces).

Upstream hashing for signatures. In signing the transcript for authentication, both `draft-05` and `draft-dh` have the signer input the *hash* of the current transcript to the signing algorithm; if the signature algorithm is a hash-then-sign algorithm, it will then perform an additional hash. From a cryptographic point of view, it would be preferable to insert the full (unhashed) transcript and let the signing algorithm opaquely take care of processing this message. For engineering purposes, however, it may be amenable to hash the transcript iteratively, only storing the intermediate values instead of entire transcript. Furthermore, since the hashed transcript is likewise given to the key derivation function, storing the hash value may be also advantageous in this regard. In our security proof, this upstream hashing leads to an additional assumption about the collision resistance of the hash function (which would otherwise be taken care of by the signature scheme).

3.2 0-RTT Handshake Mode

One of the main design goals for TLS 1.3 [Int] is to reduce handshake latency and in particular include a zero round-trip time (0-RTT) mode for handshakes with a previously seen server. While the mainline draft in version `draft-05` [Res15a] does not contain a specification for a 0-RTT handshake mode (yet), the forked `draft-dh` draft [Res15c] provisions for a 0-RTT ("early application") traffic key in its handshake key schedule. However, also the `draft-dh` version (at the time of writing) lacks a detailed specification how 0-RTT data transfer will be implemented.

Most importantly, there is still an ongoing discussion on the IETF TLS working group mailing list[6] about the handling (and potential prevention) of replay attacks, in which a man-in-the-middle attacker

[6] https://www.ietf.org/mail-archive/web/tls/current/msg15594.html and follow-up (March 2015)

replays a client's `ClientHello` and `ClientKeyShare` messages in order to make the server derive the same (0-RTT) key twice. Replay attacks on 0-RTT keys (to which only the client contributes) can, in principle, only be prevented by requiring the server to keep a register of all client key shares (e.g., the nonces) seen and reject any connections reusing a key share.

This approach is most prominently employed in Google's QUIC protocol [Ros13, LC13], using time and server-specific nonce prefixes to render the nonce register manageable in storage. The current discussion on 0-RTT replay protection in TLS 1.3 however tends to refraining from the requirement of a globally and temporally consistent server state in order to make replays always detectable. This would then, in turn, require to give up the anti-replay protection for 0-RTT data sent over TLS, an option for which the consequences with respect to implementation and application interface are still being discussed.

Missing sufficient details on design and implementation of the envisioned 0-RTT key exchange mode and data transfer in TLS 1.3, we have to omit an analysis of this protocol aspect in this work. Nevertheless, we remark that allowing replay in deriving 0-RTT keys inherently conflicts with the classical notions of session partnering established in key exchange models. Exploring the security provided for 0-RTT keys when replay is explicitly possible in a key exchange protocol as well as analyzing the yet to be defined approach in TLS 1.3 hence remains as a task for future work.

Another point of consideration for 0-RTT would be the possibility of small-subgroup-style attacks on the semi-static Diffie–Hellman exponent s. By sending many different small-subgroup generators X to be used by the server to compute a shared 0-RTT, an attacker could observe handshake messages derived from X^s. After observing a large enough number of samples, the attacker could compute $s \mod |\langle X \rangle|$ for many different $|\langle X \rangle|$, and then use the Chinese Remainder Theorem to hopefully recover s. For non–0-RTT `draft-dh` handshakes, the semi-static exponent s plays no role in our security analysis. However, this is expected to change for 0-RTT handshakes, where the semi-static secret will become relevant for security.

4 Multi-Stage Key Exchange Model

In this section we recap and extend the model for *multi-stage key exchange* by Fischlin and Günther [FG14] based on the Bellare–Rogaway-style model of Brzuska et al. [BFWW11, Brz13].

4.1 Outline of the Model for Multi-Stage Key Exchange

Our model for multi-stage key exchange protocols follows the Bellare–Rogaway paradigm [BR94]. We assume that an adversary controls the network which connects multiple sessions of honest parties, enabling the adversary to modify, inject, or drop transmissions of these honest parties. This is captured via a NewSession (for starting a new session of an honest party) and a Send query (delivering some message to it). Since the goal is to ultimately provide secrecy of the various session keys, the adversary may pose Test queries for some stage to either receive the corresponding session key of that stage, or to get an independent random key instead. Since a session key may be used to derive the next one, we need to be careful when such a Test query is admissible.

Our model allows the adversary to learn certain secret inputs to the protocol execution, as well as outputs such as session keys; we do not allow the adversary to learn intermediate values from protocol execution, as we do not aim to capture implementation flaws within the protocol. The Corrupt query models leakage of long-term authentication keys. The Reveal query models leakage of session keys. For both of these, we must prohibit compromise of secrets that make it trivial to break the security property: we do so by defining partners via session identifiers. In the multi-stage setting, each stage involves its own identifier. An important aspect for the Reveal query in the multi-stage setting concerns the security of future session keys of later stages, given that a session key of some stage is revealed. (Session-)key independence says that such leakage does not endanger future keys. Our model does not consider leakage

of `draft-dh`'s semi-static keys: since `draft-dh` does not actually include 0-RTT session keys, the leakage of semi-static secrets does not affect the security of the handshake. However, in a future protocol using semi-static secrets to derive 0-RTT session keys, security would depend on semi-static secrets and leakage would have to be modeled appropriately.

For TLS 1.3 some adaptations of the multi-stage model of Fischlin and Günther [FG14] are necessary or beneficial. In order to cover the various authenticity properties of the TLS 1.3 handshake, we extend their model to encompass, besides mutually and unilaterally authenticated keys, also unauthenticated keys. One can imagine TLS 1.3 as being composed of various protocol versions which share joint steps, but are fundamentally different in terms of security. Namely, TLS 1.3 can be seen as a family of three protocols, one without any authentication, one for unilateral authentication (of the server), and another one for mutual authentication where both client and server authenticate. We capture this by allowing the adversary in the security model to determine the type of authentication, and thus the corresponding sub protocol, when initializing a session. We also capture keys and executions with increasing authenticity properties, starting with an unauthenticated session key and then establishing a unilaterally or mutually authenticated key. We also allow executions of different types to run *concurrently*, even within a single party.

We additionally allow the communication partner of a session to be unknown at the start of the protocol, i.e., we allow for "post-specified peers" as introduced by Canetti and Krawczyk [CK02]. In our model, this is captured by letting the adversary initialize a session with a wildcard '$*$' as the intended communication partner and corresponds to the regular case in TLS 1.3 that parties discover their peer's identity during protocol execution when they receive their peer's certificate. Note that the common approach to authenticate clients by password-based login over the already established TLS connection is beyond the scope of this paper; from the perspective of our key exchange model, those are sessions where the client does *not* authenticate.

Another change concerns stronger key secrecy properties for sessions communicating with unauthenticated partners. For example, in TLS 1.3 a server can communicate with an unauthenticated client. Since the adversary could easily impersonate the unauthenticated client and thereby legitimately compute the shared session key, we cannot in general allow all server sessions with unauthenticated partners to be tested. However, if there is an *honest* unauthenticated client, then the key between these honest parties should still be secure, so we allow Test queries for sessions with unauthenticated partners *if an honest partner exists* (as done in [FG14]).

This, though, turns out to be overly restrictive and less handy for our composition result. Intuitively, one should also allow to Test such a server session even if the adversary does not deliver the server's final message to the honest client session. Since the client at this point has already completed his contribution to the session key on the server side, this key should already be considered secure. We hence introduce the notion of *contributive identifiers*, identifying sessions of honest parties which are currently not partnered according to (full) session identifiers, but indicating that the key is entirely based on an honest peer's contribution. For soundness we assume that partnered sessions (having matching session identifiers) also agree on the contributive identifier. Both session identifiers and contributive identifiers are set primarily as administrative tokens by the key exchange protocol during the execution. In contrast to session identifiers, a contributive identifier can be updated several times instead of being set only once, e.g., to eventually match the session identifier. Guidance for how and when to set contributive identifiers can be obtained by considering composition: we will show that secure usage of an established session key in a subsequent symmetric protocol is possible whenever the parties honestly (or authentically) contributed to that key, i.e., agree on the contributive identifiers. Contributive identifiers may be seen as the identifier-based analogue to prefix-matching definitions used in ACCE models [JKSS12], allowing the adversary to issue Test queries to sessions that are non-trivial to break but normally force the adversary to lose the game.

4.2 Preliminaries

We denote by $\mathcal{U}$ the set of *identities* used to model the participants in the system, each identified by some $U \in \mathcal{U}$ and associated with a certified long-term public key pk_U and secret key sk_U. Note that in addition to the long-term keys parties may also hold (uncertified) temporary ("semi-static" in `draft-dh`) key pairs for the 0-RTT protocol version, each identified by a key identifier kid. Sessions of a protocol are uniquely identified (on the administrative level of the model) using a *label* $\mathsf{label} \in \mathsf{LABELS} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$, where (U, V, k) indicates the k-th local session of identity U (the session *owner*) with V as the intended communication *partner*.

For each session, a tuple with the following information is maintained as an entry in the *session list* List_S, where values in square brackets [] indicate the default/initial value. Some variables have values for each stage $i \in \{1, \ldots, M\}$.[7]

- $\mathsf{label} \in \mathsf{LABELS}$: the (administrative) session label
- $U \in \mathcal{U}$: the session owner
- $V \in (\mathcal{U} \cup \{*\})$: the intended communication partner, where the distinct wildcard symbol '$*$' stands for "unknown identity" and can be set to a specific identity in $\mathcal{U}$ *once* by the protocol
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: the session owner's role in this session
- $\mathsf{auth} \in \mathsf{AUTH} \subseteq \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}^\mathsf{M}$: the aspired authentication type of each stage from the set of supported properties AUTH, where M is the maximum stageand auth_i indicates the authentication level in stage $i > 0$
- kid_U: the key identifier for the temporary public/secret key pair $(\mathsf{tpk}, \mathsf{tsk})$ used by the session owner
- kid_V: the key identifier for the communication partner
- $\mathsf{st}_\mathsf{exec} \in (\mathsf{RUNNING} \cup \mathsf{ACCEPTED} \cup \mathsf{REJECTED})$: the state of execution $[\mathsf{running}_0]$, where $\mathsf{RUNNING} = \{\mathsf{running}_i \mid i \in \mathbb{N}_0\}$, $\mathsf{ACCEPTED} = \{\mathsf{accepted}_i \mid i \in \mathbb{N}\}$, $\mathsf{REJECTED} = \{\mathsf{rejected}_i \mid i \in \mathbb{N}\}$
- $\mathsf{stage} \in \{0, \ldots, \mathsf{M}\}$: the current stage [0], where stage is incremented to i when $\mathsf{st}_\mathsf{exec}$ reaches $\mathsf{accepted}_i$ resp. $\mathsf{rejected}_i$
- $\mathsf{sid} \in (\{0,1\}^* \cup \{\perp\})^\mathsf{M}$: sid_i $[\perp]$ indicates the session identifier in stage $i > 0$
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\perp\})^\mathsf{M}$: cid_i $[\perp]$ indicates the contributive identifier in stage $i > 0$
- $\mathsf{K} \in (\{0,1\}^* \cup \{\perp\})^\mathsf{M}$: K_i $[\perp]$ indicates the established session key in stage $i > 0$
- $\mathsf{st}_\mathsf{key} \in \{\mathsf{fresh}, \mathsf{revealed}\}^\mathsf{M}$: $\mathsf{st}_{\mathsf{key},i}$ [fresh] indicates the state of the session key in stage $i > 0$
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^\mathsf{M}$: test indicator tested_i [false], where true means that K_i has been tested

By convention, if we add a partly specified tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, \mathsf{kid}_U, \mathsf{kid}_V)$ to List_S, then the other tuple entries are set to their default value. As labels are unique, we write as a shorthand, e.g., $\mathsf{label}.\mathsf{sid}$ for the element sid in the tuple with label label in List_S, and analogously for other entries.

4.3 Authentication Types

We distinguish between three different levels of authentication for the keys derived in a multi-stage key exchange protocol: *unauthenticated* stages and keys (which provides no authentication for either communication partner); *unilaterally authenticated* stages and keys (which authenticates one party, in our case the responder); and *mutually authenticated* stages and keys (which authenticates both communication partners). We let the adversary choose the authentication type for each session it creates.

For stages with unilateral authentication, where only the responder authenticates, we consequently only aim for secrecy of the initiator's session key, or of the responder's key, if the initiator's contribution to the key is honest and the adversary merely observes the interaction. In the non-authenticated case

[7] We fix a maximum stage M only for ease of notation. Note that M can be arbitrary large in order to cover protocols where the number of stages is not bounded a-priori.

we only ask for secrecy of those keys established through contributions of two honest parties. Since the adversary can trivially impersonate unauthenticated parties we cannot hope for key secrecy beyond that.

Formally, we capture the authenticity properties provided in a protocol by a set $\mathsf{AUTH} \subseteq \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}^{\mathsf{M}}$, representing each protocol variant's authentication by a vector $(\mathsf{auth}_1, \ldots, \mathsf{auth}_{\mathsf{M}}) \in \mathsf{AUTH}$ specifying for each stage i the protocol whether it is unauthenticated ($\mathsf{auth}_i = \mathsf{unauth}$), unilaterally authenticated ($\mathsf{auth}_i = \mathsf{unilateral}$), or mutually authenticated ($\mathsf{auth}_i = \mathsf{mutual}$). We moreover treat all authenticity variants of a protocol concurrently in our model (and hence speak about *concurrent authentication properties*): we allow concurrent executions of the different key exchange sub protocols, simultaneously covering all potential unauthenticated, unilaterally authenticated, or mutually authenticated runs. Given that the authenticity of keys is a strictly non-decreasing property with progressing stage, we also use the following simpler notation:

- **no authentication**: the keys of all stages are *unauthenticated*.
- **stage-k unilateral authentication**: the keys of stage i are *unauthenticated* for $i < k$ and *unilaterally authenticated* for $i \geq k$.
- **stage-ℓ mutual authentication**: the keys of stage i are *unauthenticated* for $i < \ell$ and *mutually authenticated* for $i \geq \ell$.
- **stage-k unilateral stage-ℓ mutual authentication**: the keys of stage i are *unauthenticated* for $i < k$, *unilaterally authenticated* for $k \leq i < \ell$, and *mutually authenticated* for $i \geq \ell$.

For example, stage-2 unilateral authentication of a three-stage key exchange protocol (one variant in TLS 1.3 `draft-05`) indicates that the first key is unauthenticated and the keys in stages 2 and 3 are unilaterally (responder-only) authenticated, formally represented by $\mathsf{auth} = (\mathsf{unauth}, \mathsf{unilateral}, \mathsf{unilateral})$. In our model, we can hence, for any specific session, tell whether a certain stage is unauthenticated, unilaterally authenticated, or mutually authenticated (from that session's point of view).

4.4 Adversary Model

We consider a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ which controls the communication between all parties, enabling interception, injection, and dropping of messages. As in [FG14] we distinguish different levels of the following three (orthogonal) security aspects of a multi-stage key exchange scheme: forward secrecy, authentication, and key dependence; the latter refers to whether the next session key relies on the confidentiality of the previous session key. Similarly to the different authentication types we also speak of *stage-k forward secrecy* if the protocol provides forward secrecy from the k-th stage onwards.

To capture admissible adversarial interactions it is convenient here to add a flag lost to the experiment which is initialized to false. This flag will later specify if the adversary loses due to trivial attacks, such as revealing the session key of a partner session to a tested session. In principle one could later check for such losing conditions when the adversary stops, but since some properties are time-critical one would need to re-capture the entire execution flow. Instead, we set the flag during the execution and oracle calls in some cases, but also check a-posteriori for other conditions.

The adversary interacts with the protocol via the following queries:

- $\mathsf{NewTempKey}(U)$: Generate a new temporary key pair $(\mathsf{tpk}, \mathsf{tsk})$, create and return a (unique) new identifier kid for it.
  Note that we do not invalidate old key identifiers of the same identity U (as protocols would presumably do), but keep the model instead as general as possible at this point, especially since active protocol runs may still rely on previous temporary keys.

- $\mathsf{NewSession}(U, V, \mathsf{role}, \mathsf{auth}, \mathsf{kid}_U, \mathsf{kid}_V)$: Creates a new session for participant identity U with role role and key identifier kid_U having V with key identifier kid_V as intended partner (potentially unspecified, indicated by $V = *$) and aiming at authentication type auth.

If there is no temporary key with identifier kid_U for user U or with identifier kid_V for user V, return the error symbol $\perp$. Otherwise, generate and return a (unique) new label label and add $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, \mathsf{kid}_U, \mathsf{kid}_V)$ to List_S.

- $\mathsf{Send}(\mathsf{label}, m)$: Sends a message m to the session with label label.
  If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, \mathsf{kid}_U, \mathsf{kid}_V, \mathsf{st}_\mathsf{exec}, \mathsf{stage}, \mathsf{sid}, \mathsf{cid}, \mathsf{K}, \mathsf{st}_\mathsf{key}, \mathsf{tested})$ in List_S, return $\perp$. Otherwise, run the protocol on behalf of U on message m and return the response and the updated state of execution $\mathsf{st}_\mathsf{exec}$. As a special case, if $\mathsf{role} = \mathsf{initiator}$ and $m = \mathsf{init}$, the protocol is initiated (without any input message).
  If, during the protocol execution, the state of execution changes to $\mathsf{accepted}_i$ for some i, the protocol execution is immediately suspended and $\mathsf{accepted}_i$ is returned as result to the adversary. The adversary can later trigger the resumption of the protocol execution by issuing a special $\mathsf{Send}(\mathsf{label}, \mathsf{continue})$ query. For such a query, the protocol continues as specified, with the party creating the next protocol message and handing it over to the adversary together with the resulting state of execution $\mathsf{st}_\mathsf{exec}$. We note that this is necessary to allow the adversary to test such a key, before it may be used immediately in the response and thus cannot be tested anymore for triviality reasons.
  If the state of execution changes to $\mathsf{st}_\mathsf{exec} = \mathsf{accepted}_i$ for some i and there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_\mathsf{exec}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_\mathsf{key}, \mathsf{tested}')$ in List_S with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{st}'_{\mathsf{key},i} = \mathsf{revealed}$, then, for key-independence, $\mathsf{st}_{\mathsf{key},i}$ is set to $\mathsf{revealed}$ as well, whereas for key-dependent security, all $\mathsf{st}_{\mathsf{key},i'}$ for $i' \geq i$ are set to $\mathsf{revealed}$. The former corresponds to the case that session keys of partnered sessions should be considered revealed as well, the latter implements that for key dependency all subsequent keys are potentially available to the adversary, too.
  If the state of execution changes to $\mathsf{st}_\mathsf{exec} = \mathsf{accepted}_i$ for some i and there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_\mathsf{exec}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_\mathsf{key}, \mathsf{tested}')$ in List_S with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{tested}'_i = \mathsf{true}$, then set $\mathsf{label}.\mathsf{K}_i \leftarrow \mathsf{label}'.\mathsf{K}'_i$ and $\mathsf{label}.\mathsf{tested}_i \leftarrow \mathsf{true}$. This ensures that, if the partnered session has been tested before, this session's key K_i is set consistently[8] and subsequent Test queries for the session here are answered accordingly.
  If the state of execution changes to $\mathsf{st}_\mathsf{exec} = \mathsf{accepted}_i$ for some i and the intended communication partner V is corrupted, then set $\mathsf{st}_{\mathsf{key},i} \leftarrow \mathsf{revealed}$.

- $\mathsf{Reveal}(\mathsf{label}, i)$: Reveals $\mathsf{label}.\mathsf{K}_i$, the session key of stage i in the session with label label.
  If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, \mathsf{kid}_U, \mathsf{kid}_V, \mathsf{st}_\mathsf{exec}, \mathsf{stage}, \mathsf{sid}, \mathsf{cid}, \mathsf{K}, \mathsf{st}_\mathsf{key}, \mathsf{tested})$ in List_S, or $i > \mathsf{stage}$, or $\mathsf{tested}_i = \mathsf{true}$, then return $\perp$. Otherwise, set $\mathsf{st}_{\mathsf{key},i}$ to $\mathsf{revealed}$ and provide the adversary with K_i.
  If there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_\mathsf{exec}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_\mathsf{key}, \mathsf{tested}')$ in List_S with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{stage}' \geq i$, then $\mathsf{st}'_{\mathsf{key},i}$ is set to $\mathsf{revealed}$ as well. This means the i-th session keys of all partnered sessions (if established) are considered revealed too.
  As above, in the case of key-dependent security, since future keys depend on the revealed key, we cannot ensure their security anymore (neither in this session in question, nor in partnered sessions). Therefore, if $i = \mathsf{stage}$, set $\mathsf{st}_{\mathsf{key},j} = \mathsf{revealed}$ for all $j > i$, as they depend on the revealed key. For the same reason, if a partnered session label' with $\mathsf{sid}_i = \mathsf{sid}'_i$ has $\mathsf{stage}' = i$, then set $\mathsf{st}'_{\mathsf{key},j} = \mathsf{revealed}$ for all $j > i$. Note that if however $\mathsf{stage}' > i$, then keys K'_j for $j > i$ derived in the partnered session are not considered to be revealed by this query since they have been accepted previously, i.e., prior to K_i being revealed in this query.

- $\mathsf{Corrupt}(U)$: Provide sk_U to the adversary. No further queries are allowed to sessions owned by U.

[8] This implicitly assumes the following property of the later-defined Match security: Whenever two partnered sessions both accept a key in some stage, these keys will be equal.

In the non-forward-secret case, for each session label owned by U and all $i \in \{1, \ldots, \mathsf{M}\}$, set $\mathsf{label.st}_{\mathsf{key},i}$ to $\mathsf{revealed}$. In this case, all (previous and future) session keys are considered to be disclosed.
In the case of stage-j forward secrecy, $\mathsf{label.st}_{\mathsf{key},i}$ is set to revealed only if $i < j$ or if $i > \mathsf{stage}$. This means that session keys before the j-th stage (where forward secrecy kicks in) as well as keys that have not yet been established are potentially disclosed.
Independent of the forward secrecy aspect, in the case of key-dependent security, setting the relevant key states to $\mathsf{revealed}$ for some stage i is done by internally invoking $\mathsf{Reveal}(\mathsf{label}, i)$, ignoring the response and also the restriction that a call with $i > \mathsf{stage}$ would immediately return $\perp$. This ensures that follow-up revocations of keys that depend on the revoked keys are carried out correctly.

- $\mathsf{Test}(\mathsf{label}, i)$: Tests the session key of stage i in the session with label label. In the security game this oracle is given a uniformly random test bit b_{test} as state which is fixed throughout the game.
If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, \mathsf{kid}_U, \mathsf{kid}_V, \mathsf{st}_{\mathsf{exec}}, \mathsf{stage}, \mathsf{sid}, \mathsf{cid}, \mathsf{K}, \mathsf{st}_{\mathsf{key}}, \mathsf{tested})$ in $\mathsf{List}_{\mathsf{S}}$ or if $\mathsf{label.st}_{\mathsf{exec}} \neq \mathsf{accepted}_i$, return $\perp$. If there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$, but $\mathsf{st}'_{\mathsf{exec}} \neq \mathsf{accepted}_i$, set the 'lost' flag to $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that keys can only be tested if they have just been accepted but not used yet, including ensuring any partnered session that may have already established this key has not used it.
If $\mathsf{label.auth}_i = \mathsf{unauth}$ or if $\mathsf{label.auth}_i = \mathsf{unilateral}$ and $\mathsf{label.role} = \mathsf{responder}$, but there is no tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ (for $\mathsf{label} \neq \mathsf{label}'$) in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{cid}_i = \mathsf{cid}'_i$, then set $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that having an honest contributive partner is a prerequisite for testing responder sessions in an unauthenticated or unilaterally authenticated stage and for testing an initiator session in an unauthenticated stage.[9]
If $\mathsf{label.tested}_i = \mathsf{true}$, return K_i, ensuring that repeated queries will be answered consistently.
Otherwise, set $\mathsf{label.tested}_i$ to true. If the test bit b_{test} is 0, sample $\mathsf{label.K}_i \leftarrow\!\!\$\ \mathcal{D}$ at random from the session key distribution $\mathcal{D}$. This means that we substitute the session key by a random and independent key which is also used for future deployments *within* the key exchange protocol. Moreover, if there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$, also set $\mathsf{label}'.\mathsf{K}'_i \leftarrow \mathsf{label.K}_i$ and $\mathsf{label}'.\mathsf{tested}'_i \leftarrow \mathsf{true}$ to ensure consistency in the special case that both label and label' are in state $\mathsf{accepted}_i$ and, hence, either of them can be tested first.
Return $\mathsf{label.K}_i$.

Secret compromise paradigm. We follow the paradigm of the Bellare–Rogaway model [BR94], focusing on the leakage of long-term secret inputs and session key outputs of the key exchange, but not on internal values within the execution of a session. This contrasts to some extent with the model by Canetti and Krawczyk [CK01] resp. LaMacchia et al. [LLM07] which include a "session state reveal" resp. "ephemeral secret reveal" query that allows accessing internal variables of the session execution.

In the context of the TLS 1.3 drafts, this means we consider the leakage of:

- *long-term / static secret keys* (such as the signing key of the server or client): allowed since long-term values have the potential to be compromised, and necessary to model forward secrecy; modeled by the $\mathsf{Corrupt}$ query.
- *session keys* (such as tk_{hs}, tk_{app}, RMS, and EMS): allowed since these are outputs of the session and are used outside the key exchange for encryption, later resumption, or exporting of keying material; modeled by the Reveal query.

We do not permit the leakage of:

- *ephemeral secret keys* (such as the randomness in the signature algorithm or the Diffie–Hellman exponent): disallowed since TLS is not designed to be secure if these values are compromised.

[9] Note that $\mathsf{List}_{\mathsf{S}}$ entries are only created for honest sessions, i.e., sessions generated by $\mathsf{NewSession}$ queries.

- *internal values / session state* (such as ES, SS, master secrets): disallowed since TLS is not designed to be secure if these values are compromised.
- *semi-static secret keys* (such as s in `draft-dh`): disallowed in the model in this paper because the security of the full `draft-dh` handshake does not depend on the secrecy of s; however, an analysis of the 0-RTT handshake should include a consideration of the leakage of semi-static secret keys.

4.5 Security of Multi-Stage Key Exchange Protocols

The security properties for multi-stage key exchange protocols are split in two games, following Fischlin et al. [FG14] and Brzuska et al. [BFWW11, Brz13]. On the one hand, Match security ensures that the session identifiers sid effectively match the partnered sessions. On the other hand, Multi-Stage security ensures Bellare–Rogaway-like key secrecy.

4.5.1 Match Security

Our notion of Match security—extended beyond [FG14] to cover different levels of key authenticity and soundness of the newly introduced contributive identifiers—ensures that the session identifiers sid effectively match the partnered sessions which must share the same view on their interaction in the sense that

1. sessions with the same session identifier for some stage hold the same key at that stage,
2. sessions with the same session identifier for some stage agree on that stage's authentication level,
3. sessions with the same session identifier for some stage share the same contributive identifier at that stage,
4. sessions are partnered with the intended (authenticated) participant,
5. session identifiers do not match across different stages, and
6. at most two sessions have the same session identifier at any stage.

The Match security game $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$ thus is defined as follows.

Definition 4.1 (Match security)**.** *Let* KE *be a key exchange protocol and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 4.4 in the following game* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$*:*

Setup. *The challenger generates long-term public/private-key pairs for each participant* $U \in \mathcal{U}$*.*

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys and has access to the queries* NewSession*,* Send*,* NewTempKey*,* Reveal*, and* Corrupt*.*

Stop. *At some point, the adversary stops with no output.*

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1$*, if at least one of the following conditions hold:*

1. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*,* $\mathsf{label.st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$*, and* $\mathsf{label'.st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$*, but* $\mathsf{label.K}_i \neq \mathsf{label'.K}_i$*. (Different session keys in some stage of partnered sessions.)*
2. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*, but* $\mathsf{label.auth}_i \neq \mathsf{label'.auth}_i$*. (Different authentication types in some stage of partnered sessions.)*
3. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*, but* $\mathsf{label.cid}_i \neq \mathsf{label'.cid}_i$ *or* $\mathsf{label.cid}_i = \mathsf{label'.cid}_i = \bot$*. (Different or unset contributive identifiers in some stage of partnered sessions.)*
4. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*,* $\mathsf{label.auth}_i = \mathsf{label'.auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$*,* $\mathsf{label.role} = \mathsf{initiator}$*, and* $\mathsf{label'.role} = \mathsf{responder}$*, but* $\mathsf{label}.V \neq \mathsf{label'}.U$ *or (only if* $\mathsf{label.auth}_i = \mathsf{mutual}$*)* $\mathsf{label}.U \neq \mathsf{label'}.V$*. (Different intended authenticated partner.)*
5. *There exist two (not necessarily distinct) labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_j \neq \bot$ *for some stages* $i, j \in \{1, \ldots, \mathsf{M}\}$ *with* $i \neq j$*. (Different stages share the same session identifier.)*

6. *There exist three distinct labels* label, label', label'' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i = \mathsf{label}''.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$. *(More than two sessions share the same session identifier.)*

We say KE *is* Match-secure *if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1\right].$$

4.5.2 Multi-Stage Security

The security game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$ defines Bellare–Rogaway-like key secrecy in the multi-stage setting as follows.

Definition 4.2 (Multi-Stage security)**.** *Let* KE *be a key exchange protocol with key distribution* $\mathcal{D}$ *and authenticity properties* AUTH*, and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 4.4 within the following game* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$*:*

Setup. *The challenger generates long-term public/private-key pairs for each participant* $U \in \mathcal{U}$*, chooses the test bit* $b_{\mathsf{test}} \leftarrow\!\!\$\ \{0,1\}$ *at random, and sets* $\mathsf{lost} \leftarrow \mathsf{false}$.

Query. *The adversary* $\mathcal{A}$ *receives the generated public keys and has access to the queries* NewSession, Send, NewTempKey, Reveal, Corrupt, *and* Test. *Note that such queries may set* lost *to* true.

Guess. *At some point,* $\mathcal{A}$ *stops and outputs a guess* b.

Finalize. *The challenger sets the 'lost' flag to* $\mathsf{lost} \leftarrow \mathsf{true}$ *if there exist two (not necessarily distinct) labels* label, label' *and some stage* $i \in \{1, \ldots, \mathsf{M}\}$ *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$, $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$, *and* $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$. *(Adversary has tested and revealed the key in a single session or in two partnered sessions.)*

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1$*, if* $b = b_{\mathsf{test}}$ *and* $\mathsf{lost} = \mathsf{false}$. *Note that the winning conditions are independent of key dependency, forward secrecy, and authentication properties of* KE*, as those are directly integrated in the affected (*Reveal *and* Corrupt*) queries and the finalization step of the game; for example,* Corrupt *is defined differently for non-forward-secrecy versus stage-j forward secrecy.*

We say KE *is* Multi-Stage-secure *in a key-dependent resp. key-independent and non-forward-secret resp. stage-j-forward-secret manner with concurrent authentication types* AUTH *if* KE *is* Match*-secure and for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1\right] - \frac{1}{2}.$$

5 Security of the draft-05 Handshake

We can now analyze the handshake as specified in TLS 1.3 `draft-05` [Res15a].

First, we define the session identifiers for the two stages deriving the handshake traffic key tk_{hs} and the application traffic key tk_{app} to be the unencrypted messages sent and received excluding the finished messages:

$$\begin{aligned}
\mathsf{sid}_1 &= (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ServerHello}, \texttt{ServerKeyShare}) \quad \text{and} \\
\mathsf{sid}_2 &= (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ServerHello}, \texttt{ServerKeyShare}, \texttt{EncryptedExtensions}^*, \\
&\quad\ \texttt{ServerCertificate}^*, \texttt{CertificateRequest}^*, \texttt{ServerCertificateVerify}^*, \\
&\quad\ \texttt{ClientCertificate}^*, \texttt{ClientCertificateVerify}^*).
\end{aligned}$$

Here, starred (*) components are not present in all authentication modes. We moreover capture the derivation of the resumption premaster secret RMS in a further stage 3 for which we define the session identifier to be $\mathsf{sid}_3 = (\mathsf{sid}_2, \text{“RMS”})$.

We stress that defining session identifiers over the *unencrypted* messages is necessary to obtain key-independent Multi-Stage security. Otherwise, we would need to either resort to key dependence, or guarantee that an adversary is not able to re-encrypt a sent message into a different ciphertext even if it knows the handshake traffic key tk_{hs} used (due to a Reveal query)—a property generally not to be expected from a (potentially randomized) encryption scheme.

Concerning the contributive identifiers, we let the client (resp. server) on sending (resp. receiving) the `ClientHello` and `ClientKeyShare` messages set $\mathsf{cid}_1 = (\texttt{CH}, \texttt{CKS})$ and subsequently, on receiving (resp. sending) the `ServerHello` and `ServerKeyShare` messages, extend it to $\mathsf{cid}_1 = (\texttt{CH}, \texttt{CKS}, \texttt{SH}, \texttt{SKS})$. The other contributive identifiers are set to $\mathsf{cid}_2 = \mathsf{sid}_2$ and $\mathsf{cid}_3 = \mathsf{sid}_3$ by each party on sending its respective `Finished` message.

As `draft-05`'s handshake does not involve semi-static keys (other than the parties' long-term keys) shared between multiple sessions, there are no temporary keys in the notation of our model. We can hence ignore NewTempKey queries in the following analysis.

Theorem 5.1 (Match security of `draft-05`)**.** *The* `draft-05` *full handshake is* Match*-secure: for any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-05},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|},$$

where n_s *is the maximum number of sessions,* q *is the group order, and* $|nonce| = 128$ *is the bit-length of the nonces.*

Match security follows from the way the session identifiers are chosen (to include all unencrypted messages), in particular guaranteeing that partnered sessions derive the same key, authenticity, and contributive identifiers. The given security bound takes into account the probability that two honest session choose the same nonce and group element.

Proof. We need to show the six properties of Match security.

1. *Sessions with the same session identifier for some stage hold the same session key.*
   For the first stage this follows as the session identifier contains the parties' Diffie–Hellman contributions g^x and g^y, which uniquely identify the Diffie–Hellman key, as well as all data entering the key derivation step. Hence, equal session identifiers imply that both parties compute the same handshake master secret and the same session key on the first stage. For the second and third stage note that the identifier sid_2 (and hence also sid_3) contains the full sid_1, implying that the parties have also computed the same handshake master secret. Since the key derivation for the stages 2 and 3 is only based on this secret value and data from sid_2, it follows that the session keys must be equal, too.
2. *Sessions with the same session identifier for some stage agree on the authenticity of the stage.*
   Observe that, for the first stage, the only admissible authenticity by design of TLS 1.3 is $\mathsf{auth}_1 = \mathsf{unauth}$ on which, hence, all sessions will agree. For the other stages, the exchanged messages (except for the finished messages) contained in the session identifier sid_2 (and hence also sid_3) uniquely determines the authenticity property for these stages. More precisely, according to the protocol specification, both sessions will agree on $\mathsf{sid}_2 = (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ServerHello}, \texttt{ServerKeyShare}, \texttt{EncryptedExtensions}^*)$ (where $\texttt{EncryptedExtensions}^*$ is optional) if and only if both have $\mathsf{auth}_2 = \mathsf{unauth}$. If sid_2 additionally contains `ServerCertificate` and `ServerCertificateVerify`, they agree on $\mathsf{auth}_2 = \mathsf{unilateral}$. If it moreover contains `CertificateRequest`, `ClientCertificate`, and `ClientCertificateVerify`, the sessions agree on mutual authentication. Moreover, $\mathsf{auth}_2 = \mathsf{auth}_3$ always holds hence same identifiers also imply agreement on authenticity.
3. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   This holds since the contributive identifier value is final once the session identifier is set.

4. *Sessions are partnered with the intended partner.*
   First of all observe that this case only applies to unilaterally or mutually authenticated stages, hence the stages 2 and 3 only. In TLS 1.3, the client obtains the server's identity within the `ServerCertificate` message and vice versa the server obtains the client's identity (in case of mutual authentication) within the `ClientCertificate` message. Moreover, honest clients and servers will not send a certificate attesting an identity different from their own. Hence, as both messages are contained in the session identifiers of stages 2 and 3 (in the respective authentication mode), agreement on sid_2 implies agreement on the respective partner's identity.
5. *Session identifiers are distinct for different stages.*
   This holds trivially as session identifiers monotonically grow with each stage.
6. *At most two sessions have the same session identifier at any stage.*
   Observe that the group element for the Diffie–Hellman key, as well as a random nonce, of both the initiator and the responder enter the session identifiers. Therefore, in order to have a threefold collision among session identifiers of honest parties, the third session would need to pick the same group element and nonce as one of the other two sessions. The probability that there exists such a collision can hence be bounded from above by $n_s^2 \cdot 1/q \cdot 2^{-|nonce|}$ where n_s is the maximum number of sessions, q is the group order, and $|nonce| = 128$ the nonces' bit-length. □

Theorem 5.2 (Multi-Stage security of `draft-05`)**.** *The* `draft-05` *full handshake is* Multi-Stage*-secure in a key-independent and stage-1-forward-secret manner with concurrent authentication properties* $\mathsf{AUTH} = \{(\mathsf{unauth}, \mathsf{unauth}, \mathsf{unauth}), (\mathsf{unauth}, \mathsf{unilateral}, \mathsf{unilateral}), (\mathsf{unauth}, \mathsf{mutual}, \mathsf{mutual})\}$ *(i.e., no authentication, stage-*2 *unilateral authentication, and stage-*2 *mutual authentication). Formally, for any efficient adversary* $\mathcal{A}$ *against the* Multi-Stage *security there exist efficient algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_8$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-05},\mathcal{A}} \leq 3n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2} \\ & + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_3} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_4} \\ & + n_s \cdot \Big(\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_5} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{PRF},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_8} \Big) \Big),\end{aligned}$$

where n_s *is the maximum number of sessions and* n_u *is the maximum number of users.*

If we charge the running time of the original security game to $\mathcal{A}$, then the running times of algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_8$ are essentially identical to the one of $\mathcal{A}$. This holds as these adversaries merely simulate $\mathcal{A}$'s original attack with some additional administrative steps.

Proof. First, we consider the case that the adversary makes a single Test query only. This reduces its advantage, according to a hybrid argument constructing out of $\mathcal{A}$ with multiple Test queries to an adversary $\mathcal{B}$ with a single Test query, by a factor at most $1/3n_s$ as there are three stages in each of the n_s sessions.[10] The main problem we have to deal with is that the session identifiers sid_2 for the second stages (and hence also sid_3) are not available in clear, but are partly encrypted. This impedes the consistent simulation for $\mathcal{B}$ in terms of identical answers for partnered sessions. Note, however, that we can recover the identifier if we have the first-stage session key, namely, the handshake traffic key. Session-key independence and further arguments about avoiding losing conditions allow us to reveal this key without endangering security of the second- and third-stage key. We provide the hybrid details in Appendix A.

From now on, we can speak about *the* session label tested at stage i. Furthermore the hybrid argument also provides the number of the test session and therefore label, so we can assume that we know label in advance.

[10] We can assume w.l.o.g. that $\mathcal{A}$ issues Test queries for a key only after that key was accepted.

Our subsequent security analysis separately considers the three (disjoint) cases that

A. the adversary tests a client session without honest contributive partner in the first stage(i.e., $\mathsf{label.role} = \mathsf{initiator}$ for the test session label and there exists no $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$),
B. the adversary tests a server session without honest contributive partner in the first stage(i.e., $\mathsf{label.role} = \mathsf{responder}$ and there exists no $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$), and
C. the tested session has an honest contributive partner in stage 1(i.e., there exists label' with $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$).

This allows us to bound the advantage as

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{draft\text{-}05},\mathcal{A}} \leq 3n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{client without partner}}_{\mathtt{draft\text{-}05},\mathcal{A}} \\ & + \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{server without partner}}_{\mathtt{draft\text{-}05},\mathcal{A}} \\ & + \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{test with partner}}_{\mathtt{draft\text{-}05},\mathcal{A}} \Big).\end{aligned}$$

Case A. Test Client without Partner

We first consider the case that the tested session is a client (initiator) session without honest contributive partner in the first stage. Since in the moment a client session can first be tested (i.e., on acceptance of the first key) cid_1 equals sid_1, we know that label also has no session partner in stage 1 (i.e., there is no other label' with $\mathsf{label.sid}_1 = \mathsf{label}'.\mathsf{sid}_1$). Having an honest partner in the second (or later) stage implies having also one in the first stage (as all messages in sid_1 are also contained in $\mathsf{cid}_2 = \mathsf{sid}_2$ and $\mathsf{cid}_3 = \mathsf{sid}_3$), hence the tested session must actually be without honest partner in all stages. Observe that, by the model conditions and sid_1 being set on the client side at the point where K_1 is accepted, the adversary cannot win in this case if the tested key is unauthenticated, hence we can assume that the key is responder-authenticated (i.e., $\mathsf{label.auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$). This allows us to focus on Test queries in the stages 2 and 3 according to the authentication properties AUTH provided.

We proceed in the following sequence of games. Starting from the original $\mathsf{Multi\text{-}Stage}$ game, we bound the advantage difference of adversary $\mathcal{A}$ between any two games by complexity-theoretic assumptions until we reach a game where the advantage of $\mathcal{A}$ is at most 0.

Game A.0. This initial game equals the $\mathsf{Multi\text{-}Stage}$ game with a single Test query where the adversary is, by our assumption, restricted to test a client (initiator) session without honest contributive partner in the first stage. Therefore,

$$\mathsf{Adv}^{G_{A.0}}_{\mathtt{draft\text{-}05},\mathcal{A}} = \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{client without partner}}_{\mathtt{draft\text{-}05},\mathcal{A}}.$$

Game A.1. In this game, we let the challenger abort the game if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H.
Let $\mathsf{abort}_{\mathsf{H}}$ denote the event that the challenger aborts in this case. We can bound the probability $\Pr[\mathsf{abort}_{\mathsf{H}}]$ by the advantage $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$ of an adversary $\mathcal{B}_1$ against the collision resistance of the hash function H. To this extent, $\mathcal{B}_1$ acts as the challenger in Game A.1, using its description of H to compute hash values, and running adversary $\mathcal{A}$ as a subroutine. If the event $\mathsf{abort}_{\mathsf{H}}$ occurs, $\mathcal{B}_1$ outputs the two distinct input values to H resulting in the same hash value as a collision.

Note that $\mathcal{B}_1$ perfectly emulates the attack of $\mathcal{A}$ according to $G_{A.0}$ up to the point till a hash collision occurs. As $\mathcal{B}_1$ wins if $\mathsf{abort}_{\mathsf{H}}$ is triggered, we have that $\Pr[\mathsf{abort}_{\mathsf{H}}] \leq \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$ and thus

$$\mathsf{Adv}^{G_{A.0}}_{\mathtt{draft\text{-}05},\mathcal{A}} \leq \mathsf{Adv}^{G_{A.1}}_{\mathtt{draft\text{-}05},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Game A.2. In this game, we let the challenger abort if the tested client session receives, within the `ServerCertificateVerify` message, a valid signature under the public key pk_U of some user $U \in \mathcal{U}$ such that the hash value has *not* been signed by any of the honest sessions.
Let $\mathsf{abort}_{\mathsf{Sig}}$ denote the event that the challenger aborts in this case. We bound the probability $\Pr[\mathsf{abort}_{\mathsf{Sig}}]$ of its occurrence by the advantage of an adversary $\mathcal{B}_2$ against the EUF-CMA security of the signature scheme Sig, denoted $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}$. In the reduction, $\mathcal{B}_2$ first guesses a user $U \in \mathcal{U}$ which it associates with the challenge public key pk^* in the EUF-CMA game, then generates all long-term key pairs for the other users $U' \in \mathcal{U} \setminus \{U\}$ and runs the Multi-Stage game $G_{A.1}$ for $\mathcal{A}$, including potentially an abort due to hash collisions. For any signature to generate for user U in honest sessions for a hash value, $\mathcal{B}_2$ calls its signing oracle about the hash value. When $\mathsf{abort}_{\mathsf{Sig}}$ is triggered, $\mathcal{B}_2$ outputs the signature the tested client received together with the hash value as a forgery.[11]

Since every honest session has a different session identifier than the tested client in the first stage (as the latter has no partnered session in this stage), no honest party will seek to sign the transcript value, expected by the tested client. Moreover, by the modification in Game A.1, there is no collision between any two honest evaluations of the hash function, so in particular there is none for the hash value computed by the tested client, implying that the hash value in question has not been signed by an honest party before. If $\mathcal{B}_2$ correctly guessed the user under whose public key the obtained signature verifies, that signature output by $\mathcal{B}_2$ is a valid forgery in the sense that its message was never queried to the EUF-CMA oracle before. Hence, $\mathcal{B}_2$ wins if $\mathsf{abort}_{\mathsf{Sig}}$ occurs and it has guessed the correct user amongst the set of (at most) n_u users and we have that $\Pr[\mathsf{abort}_{\mathsf{Sig}}] \leq n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}$ and thus

$$\mathsf{Adv}^{G_{A.1}}_{\texttt{draft-05},\mathcal{A}} \leq \mathsf{Adv}^{G_{A.2}}_{\texttt{draft-05},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}.$$

Finally, if Game A.2 does not abort, we are assured that an honest session outputs the signature obtained by the tested client session within the `ServerCertificateVerify` message. The signature is computed over $\mathsf{H}(\texttt{CH}, \texttt{CKS}, \texttt{SH}, \texttt{SKS}, \texttt{EE}^*, \texttt{SCRT}, \texttt{CR}^*)$, i.e., in particular contains all messages in sid_1. Hence, the tested client and the (distinct) honest session outputting the signature agree on sid_1, so also on cid_1, and are hence (contributively) partnered in the first stage.

The adversary $\mathcal{A}$ therefore cannot test a client (initiator) session without honest first-stage partner in Game A.2, resulting in the test bit b_{test} being unknown to $\mathcal{A}$ and hence

$$\mathsf{Adv}^{G_{A.2}}_{\texttt{draft-05},\mathcal{A}} \leq 0.$$

Case B. Test Server without Partner

We next consider the case that a server (responder) session is tested without honest contributive partner in stage 1. Again, this also implies that there is no honest partner in any of the other stages and, moreover, that also no other session shares the contributive identifiers for stages 2 and 3 as they include the full first-stage session identifier and thus also cid_1. By definition, the adversary in this case cannot win if the tested key is not mutually authenticated, hence we can assume it is, i.e., $\mathsf{label.auth}_i = \mathsf{mutual}$.

We proceed in the following sequence of games, similar to the first case, but now geared towards the (authenticating) client's signature over the protocol handshake.

Game B.0. This initial game equals the Multi-Stage game with a single Test query where the adversary this time is restricted to test a responder session without honest contributive partner in the first stage. Clearly again,

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-05},\mathcal{A}} \leq \mathsf{Adv}^{\text{1-Multi-Stage,server without partner}}_{\texttt{draft-05},\mathcal{A}}.$$

[11] Note that, although the `ServerCertificateVerify` message containing the signature is sent encrypted, the honest tested client is simulated by $\mathcal{B}_2$ and hence $\mathcal{B}_2$ can in particular decrypt this message.

Game B.1. As in the first case, this game aborts if any two honest sessions compute the same hash value for different inputs in any evaluation of H. Again, we can bound the probability $\Pr[\mathsf{abort}_{\mathsf{H}}]$ that this event occurs by the advantage of an adversary $\mathcal{B}_3$ against the hash function's collision resistance, constructed as in the first case, and obtain

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-05},\mathcal{A}} = \mathsf{Adv}^{G_{B.1}}_{\texttt{draft-05},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_3}.$$

Game B.2. This game, similar to first case, behaves as the previous one but aborts if the tested server session receives (this time within the `ClientCertificateVerify` message) a valid signature under some public key pk_U without an honest session outputting this signature. Analogously, we can bound the probability of this event, $\Pr[\mathsf{abort}_{\mathsf{Sig}}]$, by the EUF-CMA security of the signature scheme. The reduction $\mathcal{B}_4$ again encodes its challenge public key as a random user's key (and generates all other key pairs itself) and, in case of the $\mathsf{abort}_{\mathsf{Sig}}$ event occurs, outputs that very signature which the tested server session obtained in the `ClientCertificateVerify` message as its forgery.
As the client's signature contains all transcript messages up to `ClientCertificate`, it particularly fixes the first-stage session identifier sid_1, meaning that there cannot be a client session signing exactly the transcript value the tested server session is expecting since, otherwise, this would imply (contributive) partnering in stage 1. Furthermore, by Game B.1, no session will sign a value colliding under H with the tested server's transcript. Hence, if $\mathcal{B}_4$ correctly guesses the received signature's public key, it outputs a valid forgery and wins if $\mathsf{abort}_{\mathsf{Sig}}$ is triggered and thus

$$\mathsf{Adv}^{G_{B.1}}_{\texttt{draft-05},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.2}}_{\texttt{draft-05},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_4}.$$

Finally, Game B.2 ensures that an honest client session output the `ClientCertificateVerify` signature received by the tested server session which, in particular, makes these two sessions agree on sid_1, thus also on cid_1, and hence contributively partnered in the first stage. The adversary $\mathcal{A}$ therefore cannot test a server (initiator) session without honest contributive first-stage partner in Game B.2 anymore, which allows us to conclude that

$$\mathsf{Adv}^{G_{B.2}}_{\texttt{draft-05},\mathcal{A}} \leq 0.$$

Case C. Test with Partner

We finally analyze the case that the tested session (client or server) has an honest contributive partner in the first stage, i.e., we know there exists another label' such that $\mathsf{label}.\mathsf{cid}_1 = \mathsf{label}'.\mathsf{cid}_1$. Note that, in this third case, the Test query can be issued in any stage.

Game C.0. Again, we start with an initial game equal to the Multi-Stage game with a single Test query, but restricting the adversary to only test a session having an honest contributive partner in the first stage in order to have

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{draft-05},\mathcal{A}} = \mathsf{Adv}^{\text{1-Multi-Stage,test with partner}}_{\texttt{draft-05},\mathcal{A}}.$$

Game C.1. Our first modification is to guess a session $\mathsf{label}' \neq \mathsf{label}$ (among the at most n_s sessions in the game) and abort the game in case this session is not the honest contributive partner (in stage 1) of the tested session, i.e., we abort if $\mathsf{label}.\mathsf{cid}_1 \neq \mathsf{label}'.\mathsf{cid}_1$. Note that we can, without loss of generality, assume that $\mathcal{A}$ always issues a Test query. This reduces the adversary's advantage by a factor of at most $1/n_s$.

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{draft-05},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{C.1}}_{\texttt{draft-05},\mathcal{A}}.$$

From now on, we can speak of *the* session label' (contributively) partnered with the tested session label in stage 1, and can assume that we know label' in advance.

Game C.2. In Game C.2, we replace the premaster secret PMS derived in both the tested and, if derived equally, its (contributively) partnered session (label resp. label') by a randomly chosen group element $\widetilde{\mathrm{PMS}} = g^z$ for $z \leftarrow\!\!\$\ \mathbb{Z}_q$.[12]
If $\mathcal{A}$ is able to distinguish Game C.1 from Game C.2, we can turn its distinguishing capabilities into an adversary $\mathcal{B}_5$ against the decisional Diffie–Hellman (DDH) assumption in the group $\mathbb{G}$, where $\mathcal{B}_5$ receives values g^u, g^v and h, either $h = g^{uv}$ or h being random. In simulating the game for $\mathcal{A}$, algorithm $\mathcal{B}_5$ uses the challenge group elements g^u and g^v as a replacement for the values g^x and g^y to be sent in the `ClientKeyShare` and `ServerKeyShare` messages exchanged between the (predicted) test session and its partnered session and uses the third DDH challenge value h as the premaster secret PMS in both sessions. Finally, $\mathcal{B}_5$ outputs whatever $\mathcal{A}$ outputs.

Observe that, in case $h = g^{uv}$, this approach equals Game C.1 while, in case h is a random group element g^z for $z \leftarrow\!\!\$\ \mathbb{Z}_q$, the simulation equals Game C.2. Hence $\mathcal{B}_5$ provides a perfect simulation for $\mathcal{A}$, both for $h = g^{uv}$ and Game C.1, and for h being random in Game C.2. Most importantly, from $\mathcal{A}$'s perspective, g^u and g^v are chosen as in the real game since the (honest) tested session is contributively partnered in stage 1 with another honest session (i.e., $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$) and thus the adversary, by the way $\mathsf{label.cid}_1$ is set both when testing a client or a server, cannot have modified the transcript between these two sessions (up to the derivation of the premaster secret in the tested session). Moreover, both values are chosen independently of the values in all other sessions which the reduction can, hence, still select on its own. In particular, the latter also means that it is irrelevant if the Diffie–Hellman key appears in another execution but does not lead to PMS in that execution being replaced consistently; if the adversary notices this inconsistency then it still enables $\mathcal{B}_5$ to distinguish the cases.

Therefore, we can bound the difference in the advantage of $\mathcal{A}$ between the two games as

$$\mathsf{Adv}^{G_{C.1}}_{\texttt{draft-05},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.2}}_{\texttt{draft-05},\mathcal{A}} + \mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_5}.$$

Game C.3. In this game, we replace the handshake master secret HMS value by a uniformly random string $\widetilde{\mathrm{HMS}} \leftarrow\!\!\$\ \{0,1\}^\lambda$ in the tested and, if it is derived equally there, also in the contributively partnered session.
We can turn any adversary $\mathcal{A}$ able to distinguish this change with non-negligible probability into an adversary $\mathcal{B}_6$ against the security of the pseudorandom function PRF (defined for keys chosen at random from $\mathbb{G}$). We let $\mathcal{B}_6$ simulate Game C.2 as the challenger, except that it uses its PRF oracle for the derivation of HMS both in the tested and its (contributively) partnered session. Observe that, in case the oracle computes the PRF function, this equals Game C.2, whereas, if it computes a random function, this equals Game C.3. The simulation is sound because the premaster secret $\widetilde{\mathrm{PMS}}$, by the change in Game C.2, is a random element in $\mathbb{G}$ chosen independently of all other values in the game.

The advantage of $\mathcal{B}_6$ in the PRF security ($\mathsf{PRF\text{-}sec}$) game therefore bounds the advantage difference such that

$$\mathsf{Adv}^{G_{C.2}}_{\texttt{draft-05},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.3}}_{\texttt{draft-05},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{PRF},\mathcal{B}_6}.$$

Game C.4. Next, we substitute the pseudorandom function PRF in all evaluations in the tested and its partnered session using the (replaced) handshake master secret $\widetilde{\mathrm{HMS}}$ as key by a (lazy-sampled) random

[12] Note that if a server session is tested in stage 1, the (contributively partnered) client session has not yet derived PMS but will derive the same value if the adversary faithfully relays the server's response, in which case we use the same $\widetilde{\mathrm{PMS}}$ value in the then partnered session label'.

function. This, among other values, in particular affects the derivation of the handshake traffic key tk_{hs}, the master secret MS, and the resumption premaster secret RMS which are hereby replaced with random values $\widetilde{tk_{hs}}, \widetilde{\text{MS}}, \widetilde{\text{RMS}} \leftarrow\!\!\$\ \{0,1\}^\lambda$ in the tested session and, potentially, its partner.
Similar to the step in Game C.3, we can bound the difference in $\mathcal{A}$'s advantage introduced through this step by the security of the pseudorandom function PRF, this time defined for keys being uniformly random bit strings from $\{0,1\}^\lambda$. The reduction $\mathcal{B}_7$, analogously to the previous step, uses its PRF oracle for all evaluations of PRF under the key $\widetilde{\text{HMS}}$ in the tested and its partnered session. Depending on the oracle's behavior it again perfectly simulates either Game C.3 or Game C.4, as $\widetilde{\text{HMS}}$ is a uniformly random and independent bit string.

We can hence can infer that

$$\mathsf{Adv}^{G_{C.3}}_{\texttt{draft-05},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.4}}_{\texttt{draft-05},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_7}.$$

Observe that, in Game C.4, all values derived via an evaluation of PRF under the handshake master secret have now been replaced by *independent* uniformly distributed values, since each value is derived using a unique label as input to PRF.

Game C.5. Finally, we replace the application traffic key tk_{app}, derived using the pseudorandom function PRF keyed with the (random) master secret $\widetilde{\text{MS}}$, with a randomly chosen bit string $\widetilde{tk_{app}} \leftarrow\!\!\$\ \{0,1\}^\lambda$ in the tested session as well as in the partnered session (if the latter derives the same master secret $\widetilde{\text{MS}}$ in Game C.4).
Analogous to the two previous steps, we can bound this step by the security of PRF (again defined for keys being uniformly random bit strings). This (similar) reduction $\mathcal{B}_8$ is sound because $\widetilde{\text{MS}}$, by the change in Game C.4, is a uniformly random value. We can hence conclude that

$$\mathsf{Adv}^{G_{C.4}}_{\texttt{draft-05},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.5}}_{\texttt{draft-05},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_8}.$$

In Game C.5, the session keys $\widetilde{tk_{hs}}$ and $\widetilde{tk_{app}}$ as well as the resumption premaster secret $\widetilde{\text{RMS}}$ are now chosen independently and uniformly at random. As the response to its Test query is hence independent of the test bit b_{test}, the adversary $\mathcal{A}$ cannot distinguish whether it is given the real key or (another) independently chosen random value and thus

$$\mathsf{Adv}^{G_{C.5}}_{\texttt{draft-05},\mathcal{A}} \leq 0.$$

Combining the various bounds implied by the above sequences of game transitions yields the stated security bound. □

6 Security of the draft-dh Handshake

We now analyze the TLS 1.3 handshake variant as specified in the `draft-ietf-tls-tls13-dh-based` fork by Rescorla [Res15c].

Again, we define the session identifiers for the traffic key stages to be the unencrypted messages sent and received excluding the finished messages, where starred (*) components are not present in all authentication modes:

$$\begin{aligned}\mathsf{sid}_1 &= (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ServerHello}, \texttt{ServerKeyShare}) \quad \text{and}\\ \mathsf{sid}_2 &= (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ServerHello}, \texttt{ServerKeyShare}, \texttt{EncryptedExtensions}^*,\\ &\qquad \texttt{ServerCertificate}^*, \texttt{CertificateRequest}^*, \texttt{ServerParameters}^*,\\ &\qquad \texttt{ClientCertificate}^*, \texttt{ClientCertificateVerify}^*).\end{aligned}$$

We capture the further derived resumption master secret RMS and exporter master secret EMS in stages 3 and 4 and define the session identifier to be $\mathsf{sid}_3 = (\mathsf{sid}_2, \text{“RMS”})$ and $\mathsf{sid}_4 = (\mathsf{sid}_2, \text{“EMS”})$ which are uniquely determined by the second-stage identifier sid_2. As in Section 5, defining session identifiers over the *unencrypted* messages is again necessary to obtain key-independent Multi-Stage security.

The contributive identifiers are defined as for `draft-05`, i.e., cid_1 first contains `ClientHello` and `ClientKeyShare` (when they are sent resp. received) and is later augmented with `ServerHello` and `ServerKeyShare` when those messages are sent resp. received, whereas $\mathsf{cid}_i = \mathsf{sid}_i$ for stages $i \in \{2, 3, 4\}$, set by each party on sending its respective finished message.

The `draft-dh` version of the TLS 1.3 handshake involves semi-static keys g^s on the server side, which we represent as temporary keys in our model. In particular, the adversary is given power to generate arbitray many such keys via the NewTempKey query and can specify which temporary key to use in each session. We point out that temporary keys are *not* revealed in Corrupt queries, as we expect them to be used only in a short time frame.

Theorem 6.1 (Match security of `draft-dh`). *The* `draft-dh` *full handshake is* Match*-secure: for any efficient adversary $\mathcal{A}$ we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-dh},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|},$$

where n_s is the maximum number of sessions, q is the group order, and $|nonce| = 128$ is the bit-length of the nonces.

As all aspects of the `draft-dh` handshake relevant for Match security equal those of the `draft-05` handshake (aside from `ServerCertificateVerify` being renamed to `ServerParameters` and the added fourth session identifier which is distinct due to its “EMS” label), the proof of Theorem 5.1 applies here, too.

Theorem 6.2 (Multi-Stage security of `draft-dh`). *The* `draft-dh` *full handshake is* Multi-Stage*-secure in a key-independent and stage-1-forward-secret manner with concurrent authentication properties* $\mathsf{AUTH} = \{(\mathsf{unauth}, \mathsf{unauth}, \mathsf{unauth}, \mathsf{unauth}), (\mathsf{unauth}, \mathsf{unilateral}, \mathsf{unilateral}, \mathsf{unilateral}), (\mathsf{unauth}, \mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual})\}$ *(i.e., no authentication, stage-2 unilateral authentication, and stage-2 mutual authentication). Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_{10}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-dh},\mathcal{A}} \leq 4n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2} \\ & + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_3} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_4} \\ & + n_s \cdot \Big(\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_7} \\ & \qquad + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{HKDF.Extract},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_9} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}} \Big) \Big),\end{aligned}$$

where n_s is the maximum number of sessions and n_u is the maximum number of users.

Proof. First of all, as in the proof of Theorem 5.2 for `draft-05`, we consider the case that the adversary $\mathcal{A}$ makes a single Test query only. This reduces its advantage, based on an analogous hybrid argument, by a factor at most $1/4n_s$ as there are four stages in each of the n_s sessions. We from now on can speak about *the* session label tested at stage i, which we know in advance.

As for the `draft-05` proof, our security analysis is in the same three (disjoint) cases that

A. the adversary tests a client session without honest contributive partner in the first stage,
B. the adversary tests a server session without honest contributive partner in the first stage, and

C. the tested session has an honest contributive partner in stage 1.
We again can split the adversary's advantage along these three cases:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-dh},\mathcal{A}} \leq 4n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{client without partner}}_{\texttt{draft-dh},\mathcal{A}} \\ & + \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{server without partner}}_{\texttt{draft-dh},\mathcal{A}} \\ & + \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{test with partner}}_{\texttt{draft-dh},\mathcal{A}} \Big).\end{aligned}$$

The proof cases A and B are virtually identical to the respective cases in the proof for `draft-05` (cf. Theorem 5.2), aside from the server's signature message being called `ServerParameters` instead of `ServerCertificateVerify`. Again, the online signatures on the transcript ensure that no session accepts an authenticated key without partner in the first stage. We can hence focus on the third case.

Case C. Test with Partner

We analyze the case that the tested session (client or server) has an honest contributive partner in the first stage, i.e., we know there exists another label' such that $\mathsf{label}.\mathsf{cid}_1 = \mathsf{label}'.\mathsf{cid}_1$. This allows Test queries to be potentially issued in any of the four stages.

Game C.0. We start with an initial game equal to the $\mathsf{Multi\text{-}Stage}$ game with a single Test query, but restricting the adversary to only test a session having an honest contributive partner in the first stage in order to have

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{draft-dh},\mathcal{A}} = \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{test with partner}}_{\texttt{draft-dh},\mathcal{A}}.$$

Game C.1. Our first modification is to guess a session $\mathsf{label}' \neq \mathsf{label}$ (among the at most n_s sessions in the game) and abort the game in case this session is not an honest contributive partner (in stage 1) of the tested session, i.e., we abort if $\mathsf{label}.\mathsf{cid}_1 \neq \mathsf{label}'.\mathsf{cid}_1$. Note that we can, without loss of generality, assume that $\mathcal{A}$ always issues a Test query. This reduces the adversary's advantage by a factor of at most $1/n_s$.

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{draft-dh},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{C.1}}_{\texttt{draft-dh},\mathcal{A}}.$$

From now on, we can speak of *the* session label' (contributively) partnered with the tested session label in stage 1 and know label' in advance.

Game C.2. Next, we let the challenger abort the game if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H.
We can bound the probability of the game to be aborted by the advantage $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5}$ of an adversary $\mathcal{B}_5$ against the collision resistance of the hash function H similar to the Game A.1 in the proof of Theorem 5.2, where $\mathcal{B}_5$ outputs the two distinct input values to H resulting in the same hash value as a collision. It hence holds that

$$\mathsf{Adv}^{G_{C.1}}_{\texttt{draft-dh},\mathcal{A}} = \mathsf{Adv}^{G_{C.2}}_{\texttt{draft-dh},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5}.$$

Game C.3. In this game, we let the challenger abort if the client session amongst the tested and its (contributively) partnered session (i.e., the one having $\mathsf{role} = \mathsf{initiator}$) in a server-authenticated exchange ($\mathsf{label}.\mathsf{auth}_2 \in \{\mathsf{unilateral}, \mathsf{mutual}\}$) receives, within the `ServerParameters` message, a valid signature under the public key pk_U of some user $U \in \mathcal{U}$ such that the contained message has *not* been signed by any of the honest sessions.

Similarly to Game A.2 in the proof of Theorem 5.2, we can bound the probability that Game C.3 aborts by the advantage of an adversary $\mathcal{B}_6$ against the EUF-CMA security of the signature scheme Sig (multiplied by a factor n_u for guessing the right user U) and obtain

$$\mathsf{Adv}^{G_{C.2}}_{\texttt{draft-dh},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.3}}_{\texttt{draft-dh},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{EUF-CMA}_{Sig,\mathcal{B}_6}.$$

Since according to Game C.2 there is no collision between any two honest evaluations of the hash function, we can be sure that if the client session (among label and label') obtains a valid `ServerParameters` message, then this message originates from its partnered session (label' resp. label). Hence, in particular, both sessions in this case agree on the server-provided semi-static g^s value if such a value is sent. Observe that g^s (captured in our model as a temporary public key tpk with $\mathsf{tsk} = s$ being the corresponding temporary secret key) is always chosen honestly by the challenger; the adversary is only allowed to decide which particular value is used within a session.

Game C.4. In Game C.4, we replace the ephemeral secret ES derived in both the tested and (potentially) its contributively partnered session (label resp. label') by a randomly chosen group element $\widetilde{\mathrm{ES}} = g^z$ for $z \leftarrow\!\!\$\, \mathbb{Z}_q$.[13]
If $\mathcal{A}$ is able to distinguish this change, we can turn its distinguishing capabilities into an adversary $\mathcal{B}_7$ against the decisional Diffie–Hellman (DDH) assumption in the group $\mathbb{G}$, where $\mathcal{B}_7$ receives values g^u, g^v and h, either $h = g^{uv}$ or h being random. In simulating the game for $\mathcal{A}$, algorithm $\mathcal{B}_7$ uses the challenge group elements g^u and g^v as a replacement for the values g^x and g^y to be sent in the `ClientKeyShare` and `ServerKeyShare` messages exchanged between the tested and its (predicted) partnered session and uses the third DDH challenge value h as the ephemeral secret ES in both sessions. Observe that, in case $h = g^{uv}$, this approach equals Game C.3 while, in case h is a random group element g^z for $z \leftarrow\!\!\$\, \mathbb{Z}_q$, the simulation equals Game C.4. In case an `ServerParameters` message is sent (i.e., for unilateral or mutual authentication), $\mathcal{B}_7$ on the client side computes the static secret as $(g^u)^s$, using the exponent s it has chosen in the server session which we know both session agree on by Game C.3.

We observe that $\mathcal{B}_7$ provides a perfect simulation for $\mathcal{A}$, both for $h = g^{uv}$ and for h being random. Most importantly, from $\mathcal{A}$'s perspective, g^u and g^v are chosen as in the real game since the (honest) tested session is contributively partnered (in stage 1) with another honest session (i.e., $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$) and thus the adversary cannot have modified the transcript between these two sessions (up to the derivation of the ephemeral secret in the tested session). Moreover, both values are chosen independently of the ephemeral values and temporary keys in all other sessions which the reduction can, hence, still select on its own. This in particular implies that an adversary, being able to notice the (coincidental) appearance of the same Diffie–Hellman key in another execution, still allows $\mathcal{B}_7$ to distinguish the two cases by $\mathcal{A}$'s differing behavior.

We can hence bound the difference in the advantage of $\mathcal{A}$ between the two games as

$$\mathsf{Adv}^{G_{C.3}}_{\texttt{draft-dh},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.4}}_{\texttt{draft-dh},\mathcal{A}} + \mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_7}.$$

Game C.5. In this game, we replace the handshake master secret HMS and the master secret MS by uniform and independent random strings $\widetilde{\mathrm{HMS}}, \widetilde{\mathrm{MS}} \leftarrow\!\!\$\, \{0,1\}^\lambda$ in the tested session and, if derived there, in the partnered session. Only if AMS equals the 0 string used in the extraction of HMS, such that both derivations use the same salt value, then we instead use the same random string $\widetilde{\mathrm{MS}} = \widetilde{\mathrm{HMS}}$ for consistency reasons.

[13] As in the `draft-05` proof, the (contributively partnered) client session will not have derived ES yet if a server session is tested in stage 1. If however the adversary relays the server's response back without modification, the client derives the same value in which case we also use $\widetilde{\mathrm{ES}}$ in the then partnered session label'.

We can turn any adversary $\mathcal{A}$ able to distinguish this change with non-negligible probability into an adversary $\mathcal{B}_8$ against the security of the HKDF.Extract function which we model as a pseudorandom function (defined for keys chosen at random from $\mathbb{G}$). We let $\mathcal{B}_8$ simulate Game C.4 as the challenger, except that it uses its PRF oracle for the derivation of HMS and MS both in the tested and its partnered session. Observe that, in case the oracle computes the PRF function, this equals Game C.4, whereas, if it computes a random function, this equals Game C.5 (where the consistency stipulation ensures that the simulation complies with repeated answers of the random function oracle). The simulation is sound because the ephemeral secret $\widetilde{\text{ES}}$, by the change in Game C.4, is a random element in $\mathbb{G}$ chosen independently of all other values in the game and different salt values are used for the derivation of HMS and MS (given the case AMS $= 0$ is treated consistently as described above).

The advantage of $\mathcal{B}_8$ in the PRF security game therefore bounds the advantage difference such that

$$\mathsf{Adv}^{G_{C.4}}_{\texttt{draft-dh},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.5}}_{\texttt{draft-dh},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{HKDF.Extract},\mathcal{B}_8}.$$

Game C.6. Next, we replace the handshake traffic key tk_{hs} derived in both the tested and its partnered session by a uniformly random value $\widetilde{tk_{hs}}$.

Similar to the step in Game C.5, we can bound the difference in $\mathcal{A}$'s advantage introduced through this step by the security of the HKDF.Expand function which we model as a pseudorandom function, this time defined for keys being uniformly random bit strings from $\{0,1\}^\lambda$. The reduction $\mathcal{B}_9$, analogously to the previous step, uses its PRF oracle for the evaluations of HKDF.Expand under the key $\widetilde{\text{HMS}}$ in the tested and its partnered session. Depending on the oracles behavior it again perfectly simulates either Game C.5 or Game C.6, as $\widetilde{\text{HMS}}$ is a uniformly random and independent bit string.

We can hence can infer that

$$\mathsf{Adv}^{G_{C.5}}_{\texttt{draft-dh},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.6}}_{\texttt{draft-dh},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_9}.$$

Game C.7. Finally, we replace all HKDF.Expand evaluations using the (replaced) master secret $\widetilde{\text{MS}}$ as key in the tested and its partnered session by a (lazy-sampled) random function. This change affects the derivation of the handshake traffic key tk_{app}, the resumption master secret RMS, and the exporter master secret EMS which are hereby replaced with independent random values $\widetilde{tk_{app}}, \widetilde{\text{RMS}}, \widetilde{\text{EMS}} \leftarrow\!\!\$\ \{0,1\}^\lambda$ in the tested session.

As in the previous two steps, we can bound the difference in $\mathcal{A}$'s advantage introduced through this step by the PRF security of HKDF.Expand, again defined for keys being uniformly random bit strings from $\{0,1\}^\lambda$. To this extent, the reduction $\mathcal{B}_{10}$ as above uses its PRF oracle for all evaluations of HKDF.Expand under the key $\widetilde{\text{MS}}$ in the tested and its partnered session. Depending on the oracles behavior, this perfectly simulates either Game C.6 or Game C.7, as $\widetilde{\text{MS}}$ is a uniformly random and independent bit string and different labels are used in the derivation of tk_{app}, RMS, and EMS.

We can hence can infer that

$$\mathsf{Adv}^{G_{C.6}}_{\texttt{draft-dh},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.7}}_{\texttt{draft-dh},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}}.$$

In Game C.7, the session keys $\widetilde{tk_{hs}}$ and $\widetilde{tk_{app}}$ as well as the resumption and exporter master secrets $\widetilde{\text{RMS}}$ and $\widetilde{\text{EMS}}$ are now chosen independently and uniformly at random. As the response to its Test query is hence independent of the test bit b_{test}, the adversary $\mathcal{A}$ cannot distinguish whether it is given the real key or (another) independently chosen random value and thus

$$\mathsf{Adv}^{G_{C.7}}_{\texttt{draft-dh},\mathcal{A}} \leq 0.$$

Combining the various bounds implied by the above sequence of game transitions yields the stated security bound. □

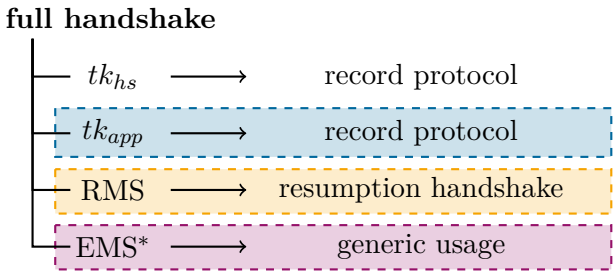

Figure 4: Illustration of the composition result applications in our analysis of the TLS 1.3 drafts. Derived keys are connected to the handshake by solid lines, their usage in protocols is indicated by an arrow. Dashed boxes indicate an application of the composition result to the usage of a specific derived (final) key in a symmetric-key protocol.
*Note that a separate exporter master secret EMS is derived only in the `draft-dh` draft.

Remark. In our analysis we do not rely on AMS to be included in the extraction step for deriving MS. In fact, Game C.5 even allows for AMS = 0 such that MS and HMS are identical. The reason why this does not harm the security is that the server's signature already authenticates the temporary key, and that all keys derived from MS resp. HMS use different input labels for the pseudorandom functions.

7 Composition

Key exchange protocols are in general of very limited use when considered on their own. Typically, such protocols are deployed as a preliminary step followed by a symmetric-key protocol (e.g., the record layer protocol in case of TLS 1.3) that makes uses of the established shared secret keys. As shown in previous work by Brzuska et al. [BFWW11] for Bellare–Rogaway-secure key exchange protocols and by Fischlin and Günther [FG14] for Multi-Stage-secure key exchange protocols, such composition can be proven to be generically secure under certain conditions.

The latter (multi-stage) result however is not yet readily applicable to the setting of TLS 1.3, as it requires the multi-stage key exchange protocol to provide—apart from key independence and forward secrecy, which TLS 1.3 satisfies—mutual authentication and a public session matching. For authentication, Fischlin and Günther state only informally how the composition theorem can be adapted to the unilateral authentication case and furthermore do not treat unauthenticated key exchange (stages). Public session matching moreover requires that, informally, an efficient algorithm eavesdropping on the communication between the adversary and the key exchange security game is able to determine the partnered sessions in the key exchange game. Since it is necessary to define session identifiers (and, hence, partnering) over the *unencrypted* messages exchanged in the TLS 1.3 handshake to achieve key independence (see Sections 5 and 6), partnering of sessions is no longer publicly decidable from the (encrypted) key exchange messages.

We therefore need to strengthen the previous composition result for multi-stage key exchange protocols [FG14] to cover, first, key exchange sessions and stages which are only unilaterally authenticated or completely unauthenticated, and, second, protocols that do not allow for a public session matching, but for one where session partnering at a certain stage i is deducible given all stage-j keys for $j < i$. Jumping ahead, knowledge of earlier stages' keys can be taken for granted as such keys can be revealed without impairing the chances of winning in a key-independent setting, which is in any case required for composition. In particular, as both achieve key independence, the analyzed TLS 1.3 handshake drafts are amenable to our composition result.

7.1 Preliminaries

Before we present our composition result we recap, partially verbatim, the syntax of composed games introduced by Brzuska et al. [BFWW11, Brz13] and extended by Fischlin and Günther [FG14] for the purpose of formal reasoning about composition of (multi-stage) key exchange and symmetric-key protocols, broadening its scope to encompass composition with *arbitrarily authenticated* multi-stage key exchange stages. Furthermore, we recap their notion of session matching and strengthen it to capture the *non–public-partnering* case.

Composed games for multi-stage key exchange. Let G_{KE} be a game modeling security for a (multi-stage) key exchange protocol KE, and G_{Π} a security game for some symmetric-key protocol Π. Fix some index i of a stage for the moment and keys derived in this stage only; the composition with protocols run on keys for other stages will follow from this via the possibility to Reveal such keys. We define $G_{\mathsf{KE}_i;\Pi}$ as the security game for the composition $\mathsf{KE}_i;\Pi$ of KE and Π where, whenever a session key K_i is accepted *in stage i* of KE where each of the two sessions involved either are authenticated or contributed honestly to the derived key[14], this key K_i is registered as a new key in the symmetric-key protocol game G_{Π}, allowing the adversary to run Π-sessions with this key (and all previously registered keys). Observe that compositional security can obviously only be guaranteed when the adversary does not know the derived session key, which we require the key exchange protocol to ensure whenever both sides of the key exchange are either authenticated or honest in their contribution. In particular, if a session key is derived in a key exchange involving an unauthenticated party whose key contribution was not simulated by the challenger, we must expect that the adversary controls this party to an extent where it holds the derived session key—and hence cannot require any security property of the symmetric-key protocol to hold for such a session key.

In $G_{\mathsf{KE}_i;\Pi}$, the adversary's task is to break the security of Π by winning in the subgame G_{Π} given access to both the queries of G_{KE} and G_{Π}, which the composed game essentially just relays to the appropriate subgame. Exceptions to this are the key registration queries of G_{Π} (that are only executed by the composed game to register stage-i keys within G_{Π} whenever such a key has been accepted), the Reveal query of G_{KE} (which the adversary is not allowed to query *for stage-i keys* in the composed game[15], as session key compromise for these keys is—if at all—captured in G_{Π}), and the Test query of G_{KE} (being only of administrative purpose for G_{KE}). The adversary wins in the composed game, if it, via its queries, succeeds in the subgame G_{Π}.

Multi-stage session matching. As established by Brzuska et al. [BFWW11], session matching is both a necessary and sufficient condition for the composition of Bellare–Rogaway-secure key exchange and generic symmetric-key protocols. They moreover observe that such a matching might not be (efficiently) computable in certain cases, e.g., if the key exchange messages are encrypted using a (publicly) re-randomizable cipher, but partnering is defined over the unencrypted messages.

The latter restriction becomes particularly relevant in the multi-stage setting, as key exchange protocols may—and TLS 1.3 does—use keys of previous stages to encrypt later stages' messages. In such cases, session matching based on the public transcript may not be feasible anymore; this especially holds for the case of TLS 1.3. We can however leverage that key independence is already a prerequisite for composition in the multi-stage setting and hence, when targeting the keys of a certain stage, revealing the keys of

[14] More formally, we consider stage-i keys which are accepted in a session label that either talks to an authenticated communication partner (i.e., $\mathsf{label.auth}_i = \mathsf{mutual}$ or $\mathsf{label.auth}_i = \mathsf{unilateral}$ and $\mathsf{label.role} = \mathsf{initiator}$) or has an honest contributing partnered session (i.e., there exists a session label' with $\mathsf{label.cid}_i = \mathsf{label}'.\mathsf{cid}_i$).

[15] Note however that keys in stages different from i, not being used for Π, are still accessible via Reveal queries in $G_{\mathsf{KE}_i;\Pi}$, which makes our result also cover *concurrent composition* with one (or several) of such protocols using the (different) keys from multiple, forward-secret stages.

previous stages is of no harm in the key exchange game. Therefore, we can strengthen session matching in the multi-stage setting to obtain also the session keys K_j for all stages $j < i$ when determining the partnering for stage i. We moreover extend session matching to comprise not only the session identifiers but also the newly introduced contributive identifiers.

Formally, we define *multi-stage session matching* as follows.

Definition 7.1 (Multi-stage session matching algorithm)**.** *A* multi-stage session matching algorithm $\mathcal{M}$ *for a key exchange protocol* KE *is an efficient algorithm for which the following holds for any adversary* $\mathcal{A}$ *playing in the* Multi-Stage *security game* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$ *of* KE*. On input a stage* i*, the public parameters of the game, an ordered list of all queries made by* $\mathcal{A}$ *and responses from* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$ *at any point of the game execution, and, for all sessions, a list of all stage-*j *keys (for any* $j < i$*) accepted at this point,* $\mathcal{M}$ *outputs two lists of pairs of all sessions in stage* i*, the first list containing exactly those pairs sharing the same session identifier* sid_i *(i.e., being partnered), and the second list exactly those pairs sharing the same contributive identifier* cid_i *at this point of the game execution.*

If such an algorithm exists for a key exchange protocol KE*, we say that* KE *allows for an* efficient multi-stage session matching*.*

7.2 Compositional Security

We can now provide our extended composition result for multi-stage key exchange: the composition $\mathsf{KE}_i;\Pi$ of a multi-stage key exchange protocol KE with an arbitrary symmetric-key protocol Π employing the stage-i session keys of KE is secure if the key exchange is Multi-Stage-secure providing *key independence*, *stage-*j *forward secrecy* (for $j \leq i$), *multi-stage session matching*, and the stage-i keys are *final*. With *final keys* in stage i (or: final stages i) we refer to those keys established after the last key exchange message has been exchanged (in TLS 1.3 this comprises keys tk_{app}, RMS, and (in `draft-dh`) EMS).[16] Note that keys derived prior to the final message exchange might be used in generating some key exchange messages and are thus not amenable to truly generic composition: such keys cannot provide security in, e.g., a symmetric-key protocol Π whose security is defined as an adversary being unable to forge a TLS 1.3 key exchange message (as an adversary can simply replay such a message from the key exchange in the composed game).[17]

Observe that we, in contrast to the previous composition result [FG14], do *not* require a particular level of authentication, but instead show compositional security for any concurrent authentication properties AUTH of KE. We remark again that, as captured in the composed game for multi-stage key exchange, security in the symmetric-key protocol Π can naturally be guaranteed only in those cases where the two parties who derived the session key are either authenticated or honestly contributed to the derived key, since otherwise we expect the adversary to know the key (e.g., by playing the role of an unauthenticated client) and cannot hope for any security.

Theorem 7.2 (Multi-stage composition)**.** *Let* KE *be a key-independent stage-*j*-forward-secret* Multi-Stage*-secure key exchange protocol with concurrent authentication properties* AUTH *and key distribution* $\mathcal{D}$ *that allows for efficient multi-stage session matching. Let* Π *be a secure symmetric-key protocol w.r.t. some game* G_Π *with a key generation algorithm that outputs keys with distribution* $\mathcal{D}$*. Then the composition* $\mathsf{KE}_i;\Pi$ *for final stages* $i \geq j$ *is secure w.r.t. the composed security game* $G_{\mathsf{KE}_i;\Pi}$*. Formally, for any efficient adversary* $\mathcal{A}$ *against* $G_{\mathsf{KE}_i;\Pi}$ *there exist efficient algorithms* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *such that*

$$\mathsf{Adv}^{G_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{B}_1} + n_s \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}_2} + \mathsf{Adv}^{G_\Pi}_{\Pi,\mathcal{B}_3},$$

[16] The notion of final keys can be formalized in our model through the sequence of special $\mathsf{Send}(\cdot, \mathsf{continue})$ queries (without further message output) at the end of a session run. A similar query can be used to enable the adversary to trigger the final key computation after the last protocol message has been sent (in TLS 1.3: after the client sent `ClientFinished`).

[17] In principle, our composition result can cover not only final, but any *unused* stage-i key. We refrain from capturing this more complex notion of non-usage of keys here.

where n_s is the maximum number of sessions in the key exchange game.

Proof of Theorem 7.2. The proof basically is an adaptation of the one for multi-stage composition by Fischlin and Günther [FG14], which in turn proceeds along the lines of the Bellare–Rogaway composition result by Brzuska et al. [BFWW11].

As a technical prerequisite, we ensure that the key exchange protocol KE in the composed game $G_{\mathsf{KE}_i;\Pi}$ always outputs the same key K_i for two partnered sessions in stage i. This basic property is given by Match security (which is subsumed under requiring $\mathsf{Multi\text{-}Stage}$ security from KE) and hence we can easily turn an adversary $\mathcal{A}$ that triggers different keys to be output in partnered sessions in the key exchange part of $G_{\mathsf{KE}_i;\Pi}$ into an adversary $\mathcal{B}_1$ against Match security. Observe that $\mathcal{B}_1$ can simply relay all oracle queries $\mathcal{A}$ makes for the KE subgame to its own oracles (note that $\mathcal{A}$ is not given access to a Test query in $G_{\mathsf{KE}_i;\Pi}$). Furthermore, $\mathcal{B}_1$ simulates the Π subgame on its own according to the $G_{\mathsf{KE}_i;\Pi}$ definition. Providing a correct simulation for $\mathcal{A}$, algorithm $\mathcal{B}_1$ always wins if $\mathcal{A}$ makes two partnered sessions output different keys in stage i; hence, we can from this point on assume that partnered sessions agree on their derived keys.

On a high level, we now first replace the derived session keys, one at a time, by a randomly chosen key from $\mathcal{D}$ and show that an adversary able to distinguish each of these replacements can be turned into an efficient $\mathsf{Multi\text{-}Stage}$ adversary against KE. After all keys have been replaced by random ones, the subgame G_Π is then independent of the key exchange protocol (as the now randomly chosen final stage-i keys are not used within the key exchange) and hence, breaking the composed game immediately translates to breaking the symmetric-key protocol game.

The first part of the proof is a hybrid argument. Let $G^{\lambda}_{\mathsf{KE}_i;\Pi}$ denote a game that behaves like $G_{\mathsf{KE}_i;\Pi}$ (with partnered sessions agreeing on the derived key), except that for the first λ accepting sessions in stage i where the key is registered in the symmetric-key protocol subgame (i.e., where the communication partner is either authenticated or an honest partnered session exists), instead of the real session key K_i a randomly chosen $\mathsf{K}'_i \leftarrow\!\!\$\ \mathcal{D}$ is registered in G_Π. Obviously, $G^0_{\mathsf{KE}_i;\Pi} = G_{\mathsf{KE}_i;\Pi}$ while $G^{n_s}_{\mathsf{KE}_i;\Pi}$ denotes the game where all keys used in the Π subgame are chosen at random from $\mathcal{D}$. Applying Lemma 7.3 below, we have that both games are indistinguishable due to the $\mathsf{Multi\text{-}Stage}$ security of KE and it holds that

$$\mathsf{Adv}^{G^0_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq \mathsf{Adv}^{G^{n_s}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} + n_s \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}_2}.$$

The main difference to the previous multi-stage composition [FG14] is that not only mutually authenticated session keys derived in the key exchange are registered in the symmetric-key protocol, but any session key for which both communication partners are either authenticated or honestly contributing. As we will see in the proof of Lemma 7.3, these cases match exactly the conditions for Test queries to be permitted in the $\mathsf{Multi\text{-}Stage}$ game which hence still allows us to replace all keys used in the symmetric-key protocol with random ones in the hybrid.

As for the previous results, in $G^{n_s}_{\mathsf{KE}_i;\Pi}$ only randomly chosen keys, independent of KE, are used in the symmetric-key protocol subgame G_Π which allows us to bound the advantage of $\mathcal{A}$ in $G^{n_s}_{\mathsf{KE}_i;\Pi}$ by the advantage of an adversary $\mathcal{B}_3$ directly breaking the protocol security game G_Π. We restate the corresponding Lemma 7.4 below without proof, as it is identically given in [FG14].

Finally, the initial assumption that Π is secure w.r.t. G_Π then allows us to conclude that $\mathsf{KE};\Pi$ is secure w.r.t. $G_{\mathsf{KE}_i;\Pi}$. □

We first establish the hybrid argument, closely following the respective proof by Fischlin and Günther [FG14].

Lemma 7.3. *Let KE be a key-independent stage-j-forward-secret $\mathsf{Multi\text{-}Stage}$-secure key exchange protocol with concurrent authentication properties AUTH and key distribution $\mathcal{D}$ that allows for an efficient multi-stage session matching and where partnered sessions in stage i always agree on the derived session key. Let*

Π *be a secure symmetric-key protocol w.r.t. some game* G_Π *with a key generation algorithm that outputs keys with distribution* $\mathcal{D}$*. Then for* $i \geq j$*, all* $\lambda = 1, \ldots, n_s$ *and any efficient adversary* $\mathcal{A}$ *there exists an efficient algorithm* $\mathcal{B}$ *such that*

$$\left| \mathsf{Adv}^{G^{\lambda-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} - \mathsf{Adv}^{G^{\lambda}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \right| \leq \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}},$$

where n_s *is the maximum number of sessions in the key exchange game.*

We provide $\mathcal{B}$ with λ as auxiliary input for simplicity but note that letting $\mathcal{B}$ pick λ at random in $[1, n_s]$ suffices to prove the hybrid argument.

Proof of Lemma 7.3. The task is to construct an algorithm $\mathcal{B}$ given λ and using the adversary $\mathcal{A}$ against $G_{\mathsf{KE}_i;\Pi}$ such that, if $\mathcal{A}$ is able to distinguish (by a a non-negligible advantage difference) between $G^{\lambda-1}_{\mathsf{KE}_i;\Pi}$ and $G^{\lambda}_{\mathsf{KE}_i;\Pi}$, then $\mathcal{B}$ has non-negligible advantage in $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$.

In order to simulate $G_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$, algorithm $\mathcal{B}$ basically forwards all KE-related queries to its Multi-Stage game as described below while answering queries to the G_Π subgame on its own (using the established stage-i keys from the key exchange). For administrative purposes, $\mathcal{B}$ keeps two mappings. The first one, $\mathsf{SDATA}\colon \mathsf{LABELS} \to \{\mathsf{initiator}, \mathsf{responder}\} \times \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\} \times [\mathcal{D}]^{i-1}$, stores the role, the i-th stage's authentication level, and the session keys for all stages $j < i$ of each key exchange session. The second one, $\mathsf{SKEY}\colon \mathsf{LABELS} \to [\mathcal{D}]$, stores the key value for each session whose stage-i key was registered in G_Π. Morever, $\mathcal{B}$ keeps a counter c, initialized as $c = 0$, indicating the number of session keys replaced by random values so far. Algorithm $\mathcal{B}$ handles queries by $\mathcal{A}$ to the key exchange subgame as follows:

- NewSession, NewTempKey, Reveal, and Corrupt queries are forwarded to $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ and the responses sent back to $\mathcal{A}$. For any NewSession call, $\mathcal{B}$ puts the issued label together with the session's specified role and authentication level for stage i into the map SDATA.
  Observe that this approach is sound and in particular does not infringe with a later test query on a stage-i key (see below). On the one hand, the stage-j forward secrecy of KE (for $j \leq i$) ensures that session keys in stage i are forward-secret and hence unaffected by Corrupt queries. On the other hand, KE being key-independent implies that $\mathsf{Reveal}(\mathsf{label}, i')$ queries allowed for stages $i' \neq i$, which are allowed in the composed game, never affect the security of session keys in stage i.

- $\mathsf{Send}(\mathsf{label}, m)$ queries are forwarded to $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ as well and the responses sent back to $\mathcal{A}$. Additionally, if session label in $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ changes to an accepting state $\mathsf{accepted}_j$ for $j < i$ due to such a query, $\mathcal{B}$ issues a query $\mathsf{Reveal}(\mathsf{label}, j)$ and stores the resulting session key K_j in the map SDATA. Note that, again by key independence, this Reveal query does not affect the session's stage-i key.
  The most important case is when session label changes to state $\mathsf{accepted}_i$. Here, $\mathcal{B}$ first of all invokes the efficient multi-stage session matching algorithm on the queries and responses $\mathcal{A}$ posed to the subgame G_{KE} and all established session keys for stages $j < i$ (which $\mathcal{B}$ has stored in SDATA), in order to obtain all sessions which are partnered and those which agree on the contributive identifier in stage i.
  In case label is partnered with some other session label' and $\mathsf{SKEY}(\mathsf{label}')$ is set, $\mathcal{B}$ sets this key value also as $\mathsf{SKEY}(\mathsf{label})$ and provides $\mathcal{A}$ with a handle for $\mathsf{SKEY}(\mathsf{label})$ in G_Π. Recall that by assumption two accepting partnered sessions always establish identical session keys (a property we ensure through Match security in the proof of Theorem 7.2).
  Otherwise, $\mathcal{B}$ checks whether the conditions for registering the resulting key in the subgame G_Π are satisfied, namely whether session label either has an authenticated communication partner (i.e., if $\mathsf{label.auth}_i = \mathsf{mutual}$ or $\mathsf{label.auth}_i = \mathsf{unilateral}$ and $\mathsf{label.role} = \mathsf{initiator}$, which $\mathcal{B}$ looks up in its

map SDATA) or has an honest contributing partnered session (i.e., the session matching outputs a session label' with $\mathsf{label}.\mathsf{cid}_i = \mathsf{label}'.\mathsf{cid}_i$). If this is the case, $\mathcal{B}$ increments the counter c and provides $\mathcal{A}$ with an identifier for $\mathsf{SKEY}(\mathsf{label})$ in G_Π, where $\mathsf{SKEY}(\mathsf{label})$ is computed depending on the counter c:
  - If $c < \lambda$, then sample $\mathsf{SKEY}(\mathsf{label}) \leftarrow\!\!\$\ \mathcal{D}$ at random.
  - If $c = \lambda$, then issue a $\mathsf{Test}(\mathsf{label}, i)$ query and store the resulting value in $\mathsf{SKEY}(\mathsf{label})$.
  - If $c > \lambda$, then issue a $\mathsf{Reveal}(\mathsf{label}, i)$ query and store the resulting value in $\mathsf{SKEY}(\mathsf{label})$.

Note that $\mathcal{B}$ first checking for partnered sessions in stage i ensures that it, if at all, only tests the first session accepting a key (avoiding the according 'lost'-flag penalty in the Test query) and never both tests and reveals a key in two partnered sessions (satisfying the finalize condition of the $\mathsf{Multi\text{-}Stage}$ definition). Moreover, as the compositional game as well as $\mathcal{B}$ only register session keys for which each of the communication partners in the key exchange is either authenticated or contributed honestly, we never test a session with an unauthenticated peer and no honest contributive partnered (satisfying the according conditions in the Test query). Therefore, $\mathcal{B}$ will never cause the 'lost' flag to be set in its $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ game.

When $\mathcal{A}$ terminates, $\mathcal{B}$ stops as well and outputs 1 if $\mathcal{A}$ has won in the composed game (i.e., in the G_Π subgame that $\mathcal{B}$ simulates on its own) and 0 otherwise. That way, if the Test query made by $\mathcal{B}$ returns the real session key, $\mathcal{B}$ perfectly simulates $G^{\lambda-1}_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$, whereas, if a random key is returned, $\mathcal{B}$ perfectly simulates $G^{\lambda}_{\mathsf{KE}_i;\Pi}$. Since $\mathcal{B}$ never causes $\mathsf{lost} = 1$ in its game we have that if $b_{\mathsf{test}} = 0$ in $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$, then $\mathcal{B}$ thus outputs the wrong bit with probability $\mathsf{Adv}^{G^{\lambda}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}$ while, if $b_{\mathsf{test}} = 1$, $\mathcal{B}$ outputs the right bit with probability $\mathsf{Adv}^{G^{\lambda-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}$. We can hence conclude that the advantage of $\mathcal{B}$ in winning the game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ is

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} \geq \left| \mathsf{Adv}^{G^{\lambda-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} - \mathsf{Adv}^{G^{\lambda}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \right| . \qquad \square$$

We complete the composition proof by restating the lemma from [FG14] that the adversary's success probability in the hybrid game $G^{n_s}_{\mathsf{KE}_i;\Pi}$, where all session keys in the G_Π subgame are chosen at random and independent of the key exchange (as the according stage i is final), can be bound by the security of the symmetric-key protocol.

Lemma 7.4. *Let* KE *be a multi-stage key exchange protocol with stage* i *being final. Let* Π *be a secure symmetric-key protocol w.r.t. some game* G_Π *with a key generation algorithm that outputs keys with distribution* $\mathcal{D}$*. Let* n_s *be the maximum number of sessions in* $G_{\mathsf{KE}_i;\Pi}$*. Then for any efficient adversary* $\mathcal{A}$ *there exists an efficient algorithm* $\mathcal{C}$ *such that*

$$\mathsf{Adv}^{G^{n_s}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq \mathsf{Adv}^{G_\Pi}_{\Pi,\mathcal{C}}.$$

8 Multi-Stage Preshared-Secret Key Exchange Model

In this section we modify the multi-stage key exchange (MSKE) framework from Section 4 to model *multi-stage preshared-secret key exchange* (MS-PSKE) security for the purpose of analyzing TLS 1.3 session resumption, obtaining a model for multi-stage key exchange protocols that use preshared keys as long-term secrets. TLS 1.3 drafts `draft-05` and `draft-dh` do not conclusively specify preshared key (PSK) ciphersuites yet, but we expect this model to be readily applicable to those as well.

In MS-PSKE, each protocol participant is identified by some $U \in \mathcal{U}$ and holds a set of pairwise preshared secrets $\mathsf{pss}_{U,V,k} = \mathsf{pss}_{V,U,k}$ (U, V, k indicating the k-th preshared secret between parties U and V)

from a fixed keyspace, associated with a unique (public) preshared-secret identifier $\mathsf{psid}_{U,V,k} = \mathsf{psid}_{V,U,k}$ and a flag $\mathsf{Corrupted}_{U,V,k}$. (For example, in TLS session resumption, the preshared-secret identifier is the `session_id` value established by the server in a field in the `ServerHello` message in the original handshake, which the client subsequently sends in its `ClientHello` message during the resumption handshake.)

Compared to our MSKE model, each entry in the session list $\mathsf{List_S}$ now contains an additional entry:

- $k \in \mathbb{N}$: the index of the preshared secret used in the protocol run between the parties U and V.

8.1 Adversary Model

Like in the MSKE model of Section 4, we consider an adversary that controls the network communication, allowing delivery, injection, modification and dropping of messages. We define a flag lost (initialized to false) that will be set to true when the adversary makes queries that would trivially break the security experiment. In the preshared secret case the common key with index k between U and V plays the role of the long-term keys and can be used to derive sessions keys in multiple (concurrent) executions, capturing many parallel session resumption steps in TLS. Corruption reveals these keys for (U, V, k) and renders all derived keys as insecure in the non-forward setting we discuss here.

The adversary interacts with the protocol via the Send, Reveal, and Test queries defined in Section 4.4, inheriting the key (in-)dependence treatment but only treating the non–forward-secret setting; our model can easily be extended to the forward-secret setting. The $\mathsf{NewSession}$ and $\mathsf{Corrupt}$ queries are modified slightly. The new query $\mathsf{NewSecret}$ allows the adversary to establish (new) preshared secrets between two parties.

- $\mathsf{NewSecret}(U, V)$: Creates a preshared secret sampled uniformly at random from the preshared secret space and stores it as $\mathsf{pss}_{U,V,k} = \mathsf{pss}_{V,U,k}$ where k is the next unused index for U and V. Also creates a unique new preshared secret identifier $\mathsf{psid}_{U,V,k} = \mathsf{psid}_{V,U,k}$ and returns $\mathsf{psid}_{U,V,k}$. Initializes $\mathsf{Corrupted}_{U,V,k}$ and $\mathsf{Corrupted}_{V,U,k}$ as fresh.
- $\mathsf{NewSession}(U, V, k, \mathsf{role}, \mathsf{auth})$: Creates a new session for party U with role role and authentication auth having V as intended partner and key index k (both V and k being potentially unspecified). A party may learn and set unspecified values during execution.
  The challenger generates a (unique) new label label and adds the entry $(\mathsf{label}, U, V, k, \mathsf{role}, \mathsf{auth})$ to $\mathsf{List_S}$.
- $\mathsf{Corrupt}(U, V, k)$: If there exists a session label with parties (U, V) or (V, U) and key identifier k and some stage i such that $\mathsf{label.tested}_i = \mathsf{true}$, then return $\perp$. Otherwise, provide the adversary with $\mathsf{pss}_{U,V,k}$ and set $\mathsf{Corrupted}_{U,V,k}$ and $\mathsf{Corrupted}_{V,U,k}$ to $\mathsf{revealed}$; in this case no further queries are allowed to sessions using $\mathsf{pss}_{U,V,k} = \mathsf{pss}_{V,U,k}$.

8.2 Security of Preshared Key Exchange Protocols

We adapt the notions for matching and multi-stage key secrecy to the preshared secret setting, essentially replacing long-term secret compromise with preshared secret compromise.

8.2.1 Match Security

As previously, Match security for preshared-secret key exchange protocols ensures that session identifiers effectively match the partnered sessions which must share the same view on their interaction. Note that the following conditions for Match security are identical to Match security conditions for MSKE models with the exception of condition 4:

1. sessions with the same session identifier for some stage hold the same key at that stage,

2. sessions with the same session identifier for some stage agree on that stage's authentication level,
3. sessions with the same session identifier for some stage share the same contributive identifier at that stage,
4. sessions are partnered with the intended (authenticated) participant, and for mutual authentication share the same key index,
5. session identifiers do not match across different stages, and
6. at most two sessions have the same session identifier at any stage.

The security game $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$ is as follows.

Definition 8.1 (Match security)**.** *Let* KE *be a key exchange protocol and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 8.1 in the following game* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$*:*

Query. *The adversary* $\mathcal{A}$ *has access to the queries* NewSecret*,* NewSession*,* Send*,* Reveal*, and* Corrupt*.*

Stop. *At some point, the adversary stops with no output.*

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1$*, if at least one of the following conditions hold:*

1. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*,* $\mathsf{label.st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$*,* $\mathsf{label'.st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$*, but* $\mathsf{label.K}_i \neq \mathsf{label'.K}_i$*. (Different session keys in the same stage of partnered sessions.)*
2. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$ *but* $\mathsf{label.auth}_i \neq \mathsf{label'.auth}_i$ *(Different authentication types in some stage of partnered sessions.)*
3. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*, but* $\mathsf{label.cid}_i \neq \mathsf{label'.cid}_i$ *or* $\mathsf{label.cid}_i = \mathsf{label'.cid}_i = \bot$*. (Different or unset contributive identifiers in some stage of partnered sessions.)*
4. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*,* $\mathsf{label.auth}_i = \mathsf{label'.auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$*,* $\mathsf{label.role} = \mathsf{initiator}$*, and* $\mathsf{label'.role} = \mathsf{responder}$*, but* $\mathsf{label}.V \neq \mathsf{label'}.U$ *or (only if* $\mathsf{label.auth}_i = \mathsf{mutual}$*)* $\mathsf{label}.U \neq \mathsf{label'}.V$ *or (only if* $\mathsf{label.auth}_i = \mathsf{mutual}$*)* $\mathsf{label}.k \neq \mathsf{label'}.k$*.*
5. *There exist two (not necessarily distinct) labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_j \neq \bot$ *for some stages* $i, j \in \{1, \ldots, \mathsf{M}\}$ *with* $i \neq j$*. (Different stages share the same session identifier.)*
6. *There exist three distinct labels* label*,* label′*,* label″ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i = \mathsf{label''.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*. (More than two sessions share the same session identifier.)*

We say KE *is* Match-secure *if for all adversaries* $\mathcal{A}$ *the following advantage is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1\right].$$

8.2.2 Multi-Stage Security

The Multi-Stage security game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$ similarly defines Bellare–Rogaway-like key secrecy in the multi-stage setting with preshared keys as follows.

Definition 8.2 (Multi-Stage security)**.** *Let* KE *be a preshared key exchange protocol with (session) key distribution* $\mathcal{D}$*, and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 8.1 in the following game* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$*:*

Setup. *Choose the test bit* $b_{\mathsf{test}} \leftarrow\!\!\$\ \{0,1\}$ *at random, and set* $\mathsf{lost} \leftarrow \mathsf{false}$*.*

Query. *The adversary has access to the queries* NewSecret*,* NewSession*,* Send*,* Reveal*,* Corrupt*, and* Test*. Note that some queries may set* lost *to* true*.*

Guess. *At some point,* $\mathcal{A}$ *stops and outputs a guess* b*.*

Finalize. *The challenger sets the 'lost' flag to* $\mathsf{lost} \leftarrow \mathsf{true}$ *if any of the following conditions hold:*

1. *There exist two (not necessarily distinct) labels* label, label' *and some stage* $i \in \{1, \ldots, \mathsf{M}\}$ *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$, $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$, *and* $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$. *(Adversary has tested and revealed the key in a single session or in two partnered sessions.)*
2. *The adversary* $\mathcal{A}$ *has issued a* $\mathsf{Test}(\mathsf{label}, i)$ *query such that* $\mathsf{Corrupted}_{\mathsf{label}.U,\mathsf{label}.V,\mathsf{label}.k} = \mathsf{revealed}$. *(Adversary has tested a session key and revealed the preshared secret used in the tested session.)*

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1$, *if* $b = b_{\mathsf{test}}$ *and* $\mathsf{lost} = \mathsf{false}$.

We say KE *is* Multi-Stage-secure *in a key-dependent/key-independent manner with concurrent authentication properties* AUTH *if* KE *is* Match*-secure and for all PPT adversaries* $\mathcal{A}$ *the following advantage is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1\right] - \frac{1}{2}.$$

9 Security of the draft-05 Session Resumption

We now turn towards session resumption and analyze the resumption handshake as specified in the `draft-ietf-tls-tls13-05` draft, denoted as d05-SR. The key schedule for resumption in `draft-dh` is not conclusively specified, so we omit a detailed analysis; since the main message flow is identical, we expect its security analysis to closely follow that of `draft-05`.

We define the session and contributive identifiers for stage 1, which derives tk_{hs}, and stage 2, which derives tk_{app}, to be both include the exchanged `ClientHello` and `ServerHello` messages as well as a distinguishing label:

$$\mathsf{sid}_1 = \mathsf{cid}_1 = (\texttt{ClientHello}, \texttt{ServerHello}, \text{“stage1”}) \quad \text{and}$$
$$\mathsf{sid}_2 = \mathsf{cid}_2 = (\texttt{ClientHello}, \texttt{ServerHello}, \text{“stage2”}).$$

By using the preshared-secret in deriving the session keys, both stages achieve mutual (implicit) authentication.

In TLS session resumption, `ClientHello` contains the field `session_id`, which serves as our preshared-secret identifier psid. This value was previously chosen by the server (the TLS standard does not specify how) and sent to the client in the `ServerHello` message in the original handshake. We assume that the `session_id` values are globally unique in TLS, for example, chosen at random from a sufficiently large space to make collisions unlikely, or of the form “*server-name* $\|$ *counter*”. We also assume each party U knows the mapping between preshared-secret identifiers $\mathsf{psid}_{U,V,k}$ and the peer identifier V and key index k for all its pre-shared secrets.

Theorem 9.1 (Match security of d05-SR)**.** *The TLS 1.3* `draft-05` *session resumption handshake* d05-SR *is* Match*-secure: for any efficient adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\text{d05-SR},\mathcal{A}} \leq n_s^2 \cdot 2^{-|nonce|},$$

where $|nonce| = 128$ *is the bitlength of the nonces.*

Proof. We need to show the six properties of Match security hold:

1. *Sessions with the same session identifier for some stage hold the same session key.*
   Recall that $\text{HMS} \leftarrow \mathsf{PRF}(\mathsf{pss}_{U,V,k}, \text{label}_1 \| \mathsf{H}(\texttt{CH}\|\texttt{SH}))$ and $\text{MS} \leftarrow \mathsf{PRF}(\text{HMS}, \text{label}_5 \| \mathsf{H}(\texttt{CH}\|\texttt{SH}))$. Moreover, $tk_{hs} \leftarrow \mathsf{PRF}(\text{HMS}, \text{label}_2 \| r_s \| r_c))$ and $tk_{app} \leftarrow \mathsf{PRF}(\text{MS}, \text{label}_6 \| r_s \| r_c)$. Since `session_id`, serving as the preshared-secret identifier psid, is a substring of the `ClientHello` message which is included in both sid_1 and sid_2, and since there is a unique mapping from psid to $\mathsf{pss}_{U,V,k}$, if two parties share the same session identifier then they both use the same PRF inputs and hence derive the same session keys.

2. *Sessions with the same session identifier for some stage agree on the authenticity of the stage.*
   This trivially holds as in TLS 1.3 `draft-05` session resumption all stages are, by definition, mutually authenticated.
3. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   This trivially holds since the contributive identifiers equal the session identifiers in each stage.
4. *Sessions are partnered with the intended partner and share the same key index.*
   Honest sessions are assured of the peer's identity and the key index via the used preshared-secret identifier psid, which is included in the session identifier for all stages; since each party knows the unique mapping between preshared-secret identifiers and key indices, a party can determine the peer's identity from the preshared-secret identifier and the mapping. Thus agreement on sid implies agreement on the partner's identity and the key index used.
5. *Session identifiers are distinct for different stages.*
   This holds trivially as session identifiers have different labels at each stage.
6. *At most two sessions have the same session identifier at any stage.*
   Both the client random and server random nonces are included in all session identifiers. To have a collision between honest parties requires two honest sessions to use the same nonces. The probability that there exists such a nonce collision is bounded by $n_s^2 \cdot 2^{-|nonce|}$ where $|nonce|$ is the length of the nonces. □

Theorem 9.2 (Multi-Stage security of d05-SR)**.** *The TLS 1.3* `draft-05` *session resumption handshake* d05-SR *is* Multi-Stage*-secure in a key-independent manner with concurrent authentication types* $\mathsf{AUTH} = \{(\mathsf{mutual}, \mathsf{mutual})\}$*: for any efficient adversary* $\mathcal{A}$ *against the* Multi-Stage *security there exist efficient algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_4$ *such that*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\text{d05-SR},\mathcal{A}} \leq 2n_s \cdot \left(\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_s \cdot \left(\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_4}\right)\right),$$

where n_s *is the maximum number of sessions.*

Proof. First, we consider the case where $\mathcal{A}$ makes a only single Test query, reducing the advantage of $\mathcal{A}$ by a factor of $1/2n_s$ (for the two stages in each of the n_s sessions) by a hybrid argument.[18] We can now focus on *the* single session with label label tested in stage i.

Proceeding in a sequence of games, we start from the original Multi-Stage game and bound the advantage difference of adversary $\mathcal{A}$ between any two games by complexity-theoretic assumptions until we reach a game where the advantage of $\mathcal{A}$ is at most 0.

Game 0. This initial game equals the Multi-Stage game with a single Test query, so

$$\mathsf{Adv}^{G_0}_{\text{d05-SR},\mathcal{A}} = \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage}}_{\text{d05-SR},\mathcal{A}}.$$

Game 1. In this game, the challenger aborts the game if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H.
Much the same as in Game A.1 of the proof of Theorem 5.2, we can break the collision resistance of H in case of this event by letting a reduction $\mathcal{B}_1$ output the two distinct input values to H. Hence:

$$\mathsf{Adv}^{G_0}_{\text{d05-SR},\mathcal{A}} \leq \mathsf{Adv}^{G_1}_{\text{d05-SR},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

[18] The hybrid argument follows the high-level idea of that in Theorem 5.2, but does not require its involved treatment of partnering as the session (and contributive) identifiers for the session resumption handshake d05-SR are public.

Game 2. As a next step, we guess the pre-shared secret pss that the tested session will use, and the challenger aborts the game if that guess was wrong. This reduces the adversary's advantage by a factor of at most $1/n_s$, thus:

$$\mathsf{Adv}^{G_1}_{\text{d05-SR},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_2}_{\text{d05-SR},\mathcal{A}}.$$

Let $\mathsf{pss}_{U,V,k}$ be the guessed pre-shared secret.

Game 3. We next replace the pseudorandom function PRF in all evaluations using the tested session's pre-shared secret $\mathsf{pss}_{U,V,k}$ as key by a (lazy-sampled) random function. This in particular affects the derivation of the handshake master secret HMS in the tested (and a potential partnered) session, which is replaced by a random value $\widetilde{\text{HMS}} \leftarrow\!\!\$\ \{0,1\}^\lambda$.
We can bound the difference this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function PRF. The according reduction $\mathcal{B}_2$ simulates Game 1, but uses its PRF oracle for any evaluation of PRF using $\mathsf{pss}_{U,V,k}$ as the key. In case the oracle computes the PRF function, this simulation equals Game 1; if it computes a random function, the simulation equals Game 3. For any successful adversary (which hence cannot invoke $\mathsf{Corrupt}$ on $\mathsf{pss}_{U,V,k}$ used in the tested session), this pre-shared key is an unknown and uniformly random value from $\mathcal{A}$'s perspective and, hence, the simulation is sound and we establish

$$\mathsf{Adv}^{G_2}_{\text{d05-SR},\mathcal{A}} \leq \mathsf{Adv}^{G_3}_{\text{d05-SR},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_2}.$$

Observe that in Game 3, the handshake master secret in the tested session (as well as its partnered session) is now a uniformly random value which is *independent* of all other values. This holds as the non-colliding (by Game 1) hash value of the `ClientHello` and `ServerHello` messages contained in each stage's session identifier is used as input to PRF, hence (even from the same pre-shared key) only partnered sessions derive the same handshake master secret.

Game 4. In this step we replace the evaluations of PRF using $\widetilde{\text{HMS}}$ as key in the tested and the potential partnered session by a (lazy-sampled) random function, thereby exchanging in particular the handshake traffic key tk_{hs} and the master secret MS with random values $\widetilde{tk_{hs}}, \widetilde{\text{MS}} \leftarrow\!\!\$\ \{0,1\}^\lambda$ (independent due to distinct input labels to PRF).
As in the previous game, we can bound the probability of $\mathcal{A}$ distinguishing this step by the security of PRF. The reducing $\mathcal{B}_3$ now uses its PRF oracle to evaluate PRF keyed with $\widetilde{\text{HMS}}$ in the tested (and partnered) session, perfectly simulating either Game 3 or Game 4 (depending on the oracle's behavior) as $\widetilde{\text{HMS}}$ is an uniformly random and independent value. Hence it holds that

$$\mathsf{Adv}^{G_3}_{\text{d05-SR},\mathcal{A}} \leq \mathsf{Adv}^{G_4}_{\text{d05-SR},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_3}.$$

Game 5. This last step exchanges the evaluations of PRF using $\widetilde{\text{MS}}$ as key (in the tested and partnered session) against a random function, leading to the application traffic key tk_{app} being replaced by a random $\widetilde{tk_{app}} \leftarrow\!\!\$\ \{0,1\}^\lambda$. Along the lines of the two previous steps we can bound this step again by the security of PRF since $\widetilde{\text{MS}}$, by Game 4, is independent and uniformly random. Therefore,

$$\mathsf{Adv}^{G_4}_{\text{d05-SR},\mathcal{A}} \leq \mathsf{Adv}^{G_5}_{\text{d05-SR},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_4}.$$

Finally, the session keys $\widetilde{tk_{hs}}$ and $\widetilde{tk_{app}}$ in Game 5 are chosen independently and uniformly at random, rendering the response to the Test query independent of the test bit b_{test}. Thus

$$\mathsf{Adv}^{G_5}_{\text{d05-SR},\mathcal{A}} \leq 0.$$

Combining the given single bounds yields the overall security statements. □

10 Conclusion

Our analyses of `draft-05` and `draft-dh` are encouraging: both TLS 1.3 candidate handshake designs can be shown to be cryptographically sound, even when restricting to standard cryptographic assumptions only. The analyses also reveal some points where the security aspects allows for flexibility and various options without endangering security, as pointed out in Section 3.

From a theoretical viewpoint, our "cascading" approach to treat session resumption not as part of the key exchange protocol, but as another symmetric-key protocol which is composed with the main handshake protocol, is useful to tame the complexity of such analyses (here and in general). Working out the details of this approach to get a full-fledged compositional analysis of the TLS 1.3 candidates is a worthwhile direction. Still, our results already confirm the sound design of the handshake protocols, as we have shown that the session keys can be safely used in the channel protocol and session resumption, and that session resumption is itself a strongly secure key exchange protocol.

Acknowledgments

We thank the anonymous reviewers for valuable comments. Marc Fischlin is supported by Heisenberg grant Fi 940/3-2 of the German Research Foundation (DFG). This work has been co-funded by the DFG as part of project S4 within the CRC 1119 CROSSING. Benjamin Dowling and Douglas Stebila are supported by Australian Research Council (ARC) Discovery Project grant DP130104304.

References

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *ACM CCS 15*, May 2015. (Cited on page 3.)

[ABP+13] Nadhem AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the security of RC4 in TLS. In *Proc. 22nd USENIX Security Symposium*, pages 305–320. USENIX, 2013. (Cited on page 3.)

[AP13] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, Berkeley, California, USA, May 19–22, 2013. IEEE Computer Society Press. (Cited on page 3.)

[AWZ10] J. Altman, N. Williams, and L. Zhu. Channel Bindings for TLS. RFC 5929 (Proposed Standard), July 2010. (Cited on page 11.)

[BBDL+15] Benjamin Beurdouche, Katheikeyan Bhargavan, Antoine Delignat-Levaud, Cedric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *Proc. IEEE Symp. on Security & Privacy (S&P) 2015*, pages 535–552. IEEE, 2015. (Cited on page 3.)

[BDF+14] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113, Berkeley, California, USA, May 18–21, 2014. IEEE Computer Society Press. (Cited on pages 3 and 11.)

[BFK+13] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing TLS with verified cryptographic security. In *2013 IEEE Symposium on Security and Privacy*, pages 445–459, Berkeley, California, USA, May 19–22, 2013. IEEE Computer Society Press. (Cited on page 5.)

[BFK+14] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella Béguelin. Proving the TLS handshake secure (as it is). In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 235–255, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on pages 4, 5, 6, and 11.)

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 51–62, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. (Cited on pages 5, 12, 18, 31, 32, and 34.)

[BMM+15] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway, and Björn Tackmann. Augmented secure channels and the goal of the TLS 1.3 record layer. Cryptology ePrint Archive, Report 2015/394, 2015. http://eprint.iacr.org/2015/394. (Cited on page 5.)

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. (Cited on pages 10, 12, and 17.)

[Brz13] Christina Brzuska. *On the Foundations of Key Exchange*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2013. http://tuprints.ulb.tu-darmstadt.de/3414/. (Cited on pages 12, 18, and 32.)

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany. (Cited on page 17.)

[CK02] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. http://eprint.iacr.org/2002/120/. (Cited on page 13.)

[Cod14] Codenomicon. The Heartbleed bug. http://heartbleed.com, April 2014. (Cited on page 3.)

[DFGS16] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas 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. https://eprint.iacr.org/2016/081. (Cited on page 3.)

[Duo11] Thai Duong. BEAST. http://vnhacker.blogspot.com.au/2011/09/beast.html, September 2011. (Cited on page 3.)

[FG14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14: 21st Conference on Computer and Communications Security*, pages 1193–1204, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. (Cited on pages 4, 5, 12, 13, 15, 18, 31, 32, 33, 34, and 36.)

[FGMP15] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 545–564, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. (Cited on page 5.)

[FKS11] Cédric Fournet, Markulf Kohlweiss, and Pierre-Yves Strub. Modular code-based cryptographic verification. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 341–350, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. (Cited on page 5.)

[Int] Internet Engineering Task Force (IETF). Charter for Transport Layer Security (TLS) Working Group. https://datatracker.ietf.org/wg/tls/charter/. (Cited on page 11.)

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. (Cited on pages 3, 4, 10, and 13.)

[Jos15] Simon Josefsson. Channel bindings for TLS based on the PRF. https://tools.ietf.org/html/draft-josefsson-sasl-tls-cb-03, March 2015. (Cited on page 11.)

[KMO+14] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Bjoern Tackmann, and Daniele Venturi. (De-)constructing TLS. Cryptology ePrint Archive, Report 2014/020, 2014. http://eprint.iacr.org/2014/020. (Cited on page 6.)

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. (Cited on pages 4 and 10.)

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. (Cited on page 9.)

[LC13] Adam Langley and Wan-Teh Chang. QUIC Crypto. https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/, June 2013. (Cited on page 12.)

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16, Wollongong, Australia, November 1–2, 2007. Springer, Heidelberg, Germany. (Cited on page 17.)

[MDK14] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This POODLE bites: Exploiting the SSL 3.0 fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf, September 2014. (Cited on page 3.)

[Res15a] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-05. https://tools.ietf.org/html/draft-ietf-tls-tls13-05, March 2015. (Cited on pages 3, 11, and 19.)

[Res15b] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-10. https://tools.ietf.org/html/draft-ietf-tls-tls13-10, October 2015. (Cited on page 3.)

[Res15c] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-dh-based. https://github.com/ekr/tls13-spec/blob/ietf92_materials/draft-ietf-tls-tls13-dh-based.txt, March 2015. (Cited on pages 3, 11, and 26.)

[Ros13] Jim Roskind. QUIC (Quick UDP Internet Connections): Multiplexed Stream Transport Over UDP. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/, December 2013. (Cited on page 12.)

A Proof of Theorem 5.2: Hybrid Argument

We provide here the details of the hybrid argument showing that if we restrict the adversary in Theorem 5.2 to a single Test query, this reduces its advantage by a factor at most $1/3n_s$ (for the three stages in each of the n_s sessions).

The hybrid argument consists of a sequence of games G_λ for $\lambda = 0, \ldots, 3n_s$, where G_λ behaves like $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{draft\text{-}05},\mathcal{A}}$ except that the first λ tested keys are the actual derived keys, and the remaining ones are replaced by random ones (uniformly chosen from $\mathcal{D}$). Here, however, we assume consistent replacements in the sense that a Test query returns the previously returned key if a partnered session has already been tested. In particular, if for a tested session there is a partner session among the first λ tested sessions, then we return the actual derived key, even if the now tested session comes after the λ's Test query. Note that, by construction, identical session identifiers yield identical keys, such that we cannot generate inconsistencies by having partners (with identical identifiers) but different keys. Also observe that G_{3n_s} equals the unmodified game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{draft\text{-}05},\mathcal{A}}$ with test bit $b_{\mathsf{test}} = 0$ even if the adversary makes less than $3n_s$ Test queries, and that in G_0 all keys are chosen uniformly at random (but consistent over partnered sessions). This means that G_0 is identical to $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{draft\text{-}05},\mathcal{A}}$ with test bit $b_{\mathsf{test}} = 1$.

For the hybrid argument we construct a reduction $\mathcal{B}$ as follows. Initially $\mathcal{B}$ chooses an index λ at random between 1 and $3n_s$. It initializes a counter c to $c = 0$ (indicating the number of tested session keys replaced by random values so far) as well as (initially empty) sets $\mathsf{SKEY}_1, \mathsf{SKEY}_2, \mathsf{SKEY}_2^e, \mathsf{SKEY}_3, \mathsf{SKEY}_3^3 \subseteq \{0,1\}^* \times [\mathcal{D}]$ for identifiers and keys in the support $[\mathcal{D}]$ of $\mathcal{D}$, to keep track of established session keys for a consistent simulation.[19] Basically, SKEY_1 corresponds to session identifier–key pairs of the first stage which $\mathcal{B}$ has already collected, similarly SKEY_2 and SKEY_3 are for the second and third stage, and SKEY_2^e resp. SKEY_3^e for transcripts and second-stage (resp. third-stage) keys where $\mathcal{B}$ cannot decrypt the data in the session identifier (yet), since the data entering the identifier are encrypted for transmission. We usually write sid_1 and $\mathsf{sid}_2 = (\mathsf{sid}_1, \mathsf{sid}_{+2})$ for the session identifiers for the first and second stage, respectively, denoting the second part of the stage-two identifier as sid_{+2}; we also write $\{\mathsf{sid}_{+2}\}$ to denote the fact that $\mathcal{B}$ only holds an encrypted version of the second part. Recall that $\mathsf{sid}_3 = (\mathsf{sid}_2, \text{“RMS”})$, hence we can

[19] As we will see, we can, by construction, always decide partnering in stage 3 (deriving the resumption master secret RMS) whenever we can decide it in stage 2. We will nevertheless state the according computations explicitly here for completeness.

also write it as $\mathsf{sid}_3 = (\mathsf{sid}_1, \mathsf{sid}_{+2}, \text{“RMS”})$. Both SKEY_1, SKEY_2, and SKEY_3 will be (individually and together) consistent during the entire simulation in the sense that they do not contain entries with identical session identifiers but different keys. This is clearly true upon initialization and remains so whenever we add elements to either set.

Algorithm $\mathcal{B}$ then runs $\mathcal{A}$, relaying all queries and answers of $\mathcal{A}$ to its external oracles, with one exception: if $\mathcal{A}$ makes one of its multiple Test queries (where we assume that all such queries are made for accepted executions only), then $\mathcal{B}$ increments c and checks c against λ.

- If $c < \lambda$ then $\mathcal{B}$ simply makes a Reveal query for the corresponding stage and returns the obtained key K. Then $\mathcal{B}$ does the following updates to its lists SKEY_1, SKEY_2, SKEY_2^e, SKEY_3, and SKEY_3^e.
  If the inspected session is a stage-one session with identifier sid_1 then $\mathcal{B}$ places the session identifier and the returned key value into SKEY_1. Then, for each element $(\mathsf{sid}'_1, \{\mathsf{sid}'_{+2}\})$ in SKEY_2^e which carries $\mathsf{sid}'_1 = \mathsf{sid}_1$ as part of the session identifier, use the session key for sid_1 to recover the (decrypted) identifier sid'_2 for the entry and put the identifier sid'_2 with its key into SKEY_2. Proceed analogously for each element $(\mathsf{sid}_{1'}, \{\mathsf{sid}'_{+2}\}, \text{“RMS”})$ in SKEY_3^e with $\mathsf{sid}'_1 = \mathsf{sid}_1$.
  If the inspected session is a stage-two session then $\mathcal{B}$ can recover the stage-two identifier $(\mathsf{sid}_1, \{\mathsf{sid}_{+2}\})$ (with some stage-one identifier sid_1 and some encrypted part). It checks if the stage-one identifier part sid_1 (in clear) is already in SKEY_1. If so,[20] then use the session key of the sid_1 entry to recover the full identifier $\mathsf{sid}_2 = (\mathsf{sid}_1, \mathsf{sid}_{+2})$ of the inspected session in clear, and put sid_2 with the key into SKEY_2. If there is no sid_1-entry in SKEY_1 then put the partly encrypted session identifier $(\mathsf{sid}_1, \{\mathsf{sid}_{+2}\})$ with the returned key into SKEY_2^e.
  For third-stage queries, proceed as for stage 2 with the according identifier $(\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“RMS”})$.
  Note that all three cases cannot introduce any inconsistencies in SKEY_1, SKEY_2, or SKEY_3 as the derived and revealed key is uniquely determined given sid_1, sid_2, resp. sid_3.
- If $c = \lambda$ then $\mathcal{B}$ proceeds as follows. Algorithm $\mathcal{B}$ first extracts the session identifier of the tested session. This is trivial for a first-stage identifier sid_1 as it consists of the communication data in clear. For a second- or third-stage identifiers $\mathsf{sid}_2 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\})$ resp. $\mathsf{sid}_2 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“RMS”})$, which also contains messages sent encrypted under the first-stage handshake traffic key, algorithm $\mathcal{B}$ will make a Reveal query for the first stage of the test session to get the handshake traffic key. It puts the corresponding session identifier sid_1 and the revealed key into SKEY_1 for future reference. Revealing this first-stage session key is admissible due to key independence, despite the tested stage-two/stage-three key; it cannot force $\mathcal{B}$ with its single Test query to lose. At the same time it allows $\mathcal{B}$ to decrypt and recover the values for sid_{+2} in clear. Algorithm $\mathcal{B}$ also checks if one can now decrypt and move any entries in SKEY_2^e to SKEY_2 resp. from SKEY_3^e to SKEY_3 (by checking for entries in SKEY_2^e resp. SKEY_3^e which carry the same stage-one identifier sid_1).
  Given that $\mathcal{B}$ now holds the session identifier (in clear) it can check if there already exists an entry in SKEY_1, SKEY_2, or SKEY_3. If so, it returns the corresponding stored key. Else, $\mathcal{B}$ uses its single external Test query to get a key K, adds this key with the recovered identifier to the corresponding set SKEY_1, SKEY_2, or SKEY_3, and returns the key to $\mathcal{A}$. Note that here SKEY_1, SKEY_2, and SKEY_3 are still consistent in any case as $\mathcal{B}$, if at all, adds a new session identifier.
- If $c > \lambda$ then $\mathcal{B}$ first recovers the session identifier of the requested test session. For a stage-one identifier sid_1 this is again easy by inspecting the communication so far. For a stage-two or stage-three identifier $\mathsf{sid}_2 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\})$ resp. $\mathsf{sid}_3 = (\mathsf{sid}_1, \{\mathsf{sid}_{+2}\}, \text{“RMS”})$ algorithm $\mathcal{B}$ first checks if there already exists an entry in SKEY_1 for the contained stage-one part sid_1 of the identifier of the inspected session and, if so, uses it to recover the full (unencrypted) sid_{+2} part of the identifier. If there is no entry then $\mathcal{B}$ makes a Reveal query for the stage-one key to again recover the full identifier sid_2 resp. sid_3 and places sid_1 and the returned key into SKEY_1. Note that such a Reveal query cannot

[20] Recall that any such entry would be unique.

infringe with $\mathcal{B}$'s single Test query, because either the Test query was for a stage-two or stage-three session (and key independence enables us to reveal any stage-one key then), or the Test query was for a stage-one identifier in which case it must already be included in SKEY_1 and the Reveal query is not made.

Given that $\mathcal{B}$ now knows the session identifier of the requested test session it checks if there is already an entry in SKEY_1, SKEY_2, or SKEY_3 for it. If so, it returns the same key as in the entry. Else it picks a key K at random from $\mathcal{D}$, returns it to $\mathcal{A}$, and adds the obtained session identifier with the key value K to the corresponding set SKEY_1, SKEY_2, or SKEY_3. Note again that there cannot exist such an entry in the lists if $\mathcal{B}$ adds some value, such that consistency remains intact.

Note that $\mathcal{B}$ provides a consistent simulation as any pair of Test queries for partnered sessions return identical answers: For Test queries of $\mathcal{A}$ for partnered sessions, both with $c < \lambda$, this is clear as the Reveal queries make $\mathcal{B}$ return consistent keys. If the second query is for $c = \lambda$ then $\mathcal{B}$ either has the identifier already in SKEY_1, SKEY_2, or SKEY_3 and answers consistently, or the values for the first Test query are at least in SKEY_2^e or SKEY_3^e and are now moved to SKEY_2 resp. SKEY_3 because $\mathcal{B}$ learns the key to sid_1 and first checks membership in SKEY_1, SKEY_2, or SKEY_3 before possibly making the Test query. If the second Test query is for $c > \lambda$ then the same argument as in the previous case applies. The latter is also true if the first Test query has been for $c = \lambda$ or for $c > \lambda$, because then the session identifier will be in SKEY_1, SKEY_2, or SKEY_3 already.

Up to the finalization step $\mathcal{B}$'s simulation is perfect (except for potentially the state of the lost flag, see below). In particular, $\mathcal{B}$ loses according to the lost flag, either set during the processing of a Test query or in the finalization step, only if $\mathcal{A}$ in the simulation (and thus in a genuine execution) would lose. Conversely, as is, it can happen that $\mathcal{B}$ even avoids a loss which $\mathcal{A}$ would trigger with a Test query for a revealed partner, but $\mathcal{B}$ omits this Test query since it provides the answer differently. This corresponds to the finalize condition of Definition 4.2. Remarkably, this causes the following problem: if $\mathcal{A}$ decides to create a difference between the two cases, genuine keys or random ones in Test queries, by deliberately losing via, say, a Reveal query for a tested partner, this difference could vanish in $\mathcal{B}$'s simulation. In order to avoid this, we let $\mathcal{B}$ eventually run the internal finalization step and check if $\mathcal{A}$ loses (and if so, forcing a loss in its simulation by making a Reveal query to the same key the Test query was issued on).

To check for the condition in the finalization step note that all Test requests of $\mathcal{A}$ insert some values in the sets $\mathsf{SKEY}_1, \mathsf{SKEY}_2$, SKEY_2^e, SKEY_3, or SKEY_3^e. Only for those entries in SKEY_2^e and SKEY_3^e algorithm $\mathcal{B}$ cannot (yet) recover the session identifier; in particular there is no entry in SKEY_1 for those values, else they would have been moved to SKEY_2 resp. SKEY_3 already. Algorithm $\mathcal{B}$ can now "clean up" the sets SKEY_2^e and SKEY_3^e and move all entries to SKEY_2 resp. SKEY_3, by making a-posteriori Reveal queries for the first-stage keys for all sessions in SKEY_2^e and SKEY_3^e to get the session key which allows us to decrypt the stage-two and stage-three identifier. These Reveal queries cannot force $\mathcal{B}$ to lose as the session identifier of the single tested session must be different (otherwise there would be an entry in SKEY_1). So we can from now on assume that $\mathcal{B}$ knows all session identifiers of $\mathcal{A}$'s requested test sessions in clear, and holds candidates for all first-stage keys of the tested sessions.

It remains that $\mathcal{B}$ checks the condition of the finalization (i.e., that $\mathcal{A}$ has not made a Reveal query to a partner of a tested session) as follows. Algorithm $\mathcal{B}$ recovers all the session identifiers of the revealed sessions (excluding the Reveal queries which only $\mathcal{B}$ made). For a stage-one Reveal request this is trivial, for a stage-two or stage-three request (with partial identifier sid_1) algorithm $\mathcal{B}$ checks if sid_1 appears among the tested sessions. If not, then this Reveal query clearly does not infringe with the Test queries. If it does appear, however, then we already have the first-stage key for sid_1 and can recover the full session identifier of the Reveal query and compare it to the set of tested sessions. If and only if $\mathcal{B}$ finds some match for some Reveal query then it forces a loss in its game.

Next, we check the losing condition within the Test query triggered when $\mathcal{A}$ requests some test such that another honest execution has already used this session key (in which case the adversary could potentially

distinguish a random key). This check is easy to perform for $\mathcal{B}$ because, in handling the Test query, it always establishes the according session identifier sid_i of the tested session's stage i. Hence, $\mathcal{B}$ can simply check whether there exists a partnered session in stage i whose execution state is already beyond $\mathsf{accepted}_i$ and force a loss in this case.

To check the condition within the Test query that $\mathcal{A}$ has not tested a session for which the partner is unauthenticated but which does not have an honest contributive partner, $\mathcal{B}$ can, for the first stage, simply inspect the the transcript as the elements of the contributive identifier cid_1 for the first stage are sent in clear. For stages 2 and 3 recall that $\mathsf{cid}_2 = \mathsf{sid}_2$ and $\mathsf{cid}_3 = \mathsf{sid}_3$ and hence $\mathcal{B}$ can again leverage the established session identifier of the tested session's stage to check if there exists an honest contributive partner for these stages upon finalization. In each case, if no contributive partnered session exists, then $\mathcal{B}$ provokes a loss.

With the final checks we have made sure that $\mathcal{B}$ loses due to some inadmissible query if and only if $\mathcal{A}$ would in the real attack. In particular, it follows that for fixed $\lambda = 0$ the simulation of $\mathcal{B}$ has exactly the same success probability as $\mathcal{A}$ in game G_0, and analogously for $\lambda = 3n_s$. A standard counting argument, basically considering the conditional probabilities for fixed choices of λ, now shows that the advantage of $\mathcal{A}$ is at most a factor $3n_s$ of the advantage of $\mathcal{B}$. More formally, noting that for some fixed λ and test bit $b_{\mathsf{test}} = 0$ we actually run the game for $b_{\mathsf{test}} = 1$ and $\lambda - 1$, we obtain:

$$
\begin{aligned}
&\Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} = 1 | b_{\mathsf{test}} = 1\right] - \Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} = 1 | b_{\mathsf{test}} = 0\right] \\
&= \frac{1}{3n_s} \cdot \sum_{\lambda_0=1}^{3n_s} \left(\Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} = 1 | \lambda = \lambda_0, b_{\mathsf{test}} = 1\right] - \Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} = 1 | \lambda = \lambda_0, b_{\mathsf{test}} = 0\right]\right) \\
&= \frac{1}{3n_s} \cdot \sum_{\lambda_0=1}^{3n_s} \left(\Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} = 1 | \lambda = \lambda_0, b_{\mathsf{test}} = 1\right] - \Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} = 1 | \lambda = \lambda_0 - 1, b_{\mathsf{test}} = 1\right]\right) \\
&= \frac{1}{3n_s} \cdot \left(\Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} = 1 | \lambda = 3n_s, b_{\mathsf{test}} = 1\right] - \Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} = 1 | \lambda = 0, b_{\mathsf{test}} = 1\right]\right) \\
&= \frac{1}{3n_s} \cdot \left(\Pr\left[G_{3n_s} = 1\right] - \Pr\left[G_0 = 1\right]\right).
\end{aligned}
$$

Noticing that the first and last differences of probabilities in both cases, for $\mathcal{B}$ and for $\mathcal{A}$, correspond to $2 \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ and $2 \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$, the claim follows.