A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates

Benjamin Dowling
Electrical Engineering and Computer Science
Queensland University of Technology
Brisbane, Australia
b1.dowling@qut.edu.au

Marc Fischlin
Cryptoplexity
Technische Universität Darmstadt
Darmstadt, Germany
marc.fischlin@cryptoplexity.de

Felix Günther
Cryptoplexity
Technische Universität Darmstadt
Darmstadt, Germany
guenther@cs.tu-darmstadt.de

Douglas Stebila
Electrical Engineering and Computer Science
Mathematical Sciences
Queensland University of Technology
Brisbane, Australia
stebila@qut.edu.au

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. This is even more important as there are two related, yet slightly different, candidates in discussion for TLS 1.3, 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.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813653.

Categories and Subject Descriptors

C.2.0 [**Computer–Communication Networks**]: General—*Security and protection*

Keywords

Transport Layer Security (TLS); key exchange; protocol analysis; composition

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 [2]), errors in the design of the TLS protocol (BEAST [16], the Lucky 13 attack [3], the triple handshake attack [8], the POODLE attack [24], the Logjam attack [1]), or flaws in implementations (the Heartbleed attack [14], state machine attacks (SMACK [7])). 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.

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` [25] (which we shorten to `draft-05`), the other one is the forked `draft-ietf-tls-tls13-dh-based` [27] (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 [19].

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 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 [19] or (b) a cunning approach to release the record layer key early [10] (see also Section 1.3). 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.

1.1 TLS 1.3 as a Multi-Stage Key Exchange

The message flow for both drafts is similar and shown in Figures 1 and 2. It is convenient to view TLS 1.3 as a *multi-stage* key exchange protocol [17] 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.

1.2 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 [17] 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 [19, 23].

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. [12] and Fischlin and Günther [17] in Section 7 to multi-stage key exchange protocols with mixed unauthenticated, unilateral, and mutual authentication.

[1]Since May 2015, two follow-up draft versions of TLS 1.3 have been published; `draft-07` [26] builds on `draft-05` but incorporates major key exchange changes including the `draft-dh` key schedule. We expect that our `draft-dh` analysis can be adapted to cover `draft-07`'s design.

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

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

A key point to secure composition of multi-stage key agreement protocols with arbitrary symmetric-key protocols in [17] 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.

Recent work by Badertscher et al. [5] shows that the authenticated encryption (with associated data) used in the record layer in both TLS 1.3 drafts is secure. Our compositional approach immediately implies that the 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.

This proceedings version omits some details due to length restrictions; additional explanation, particularly in the security model, as well as detailed proofs, appear in the full version [15].

1.3 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. [10], 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 [18, 9]. 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 analysis of Bhargavan et al. [10] is to overcome the session key usage in the finishing messages of the TLS 1.2 handshake by separating these from the remaining handshake, achieved through so-called (peer-)exchange variables to safely determine partners. While the `Finished`-message problem is eliminated in TLS 1.3, we also introduce contributive identifiers as an alternative partnering concept. In contrast to [10] those resemble closer the common session identifiers (which we also use) and are only employed for non-mutually authenticated sessions and our compositional result. We furthermore use a general composition approach to deal with resumption, while [10] considers resumption as an abbreviated handshake, executed in a different epoch. Finally, the analysis in [10] extends to the implementation level; our results are purely on the abstract level.

Concurrently to our work, Kohlweiss et al. [21] 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.

1.4 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

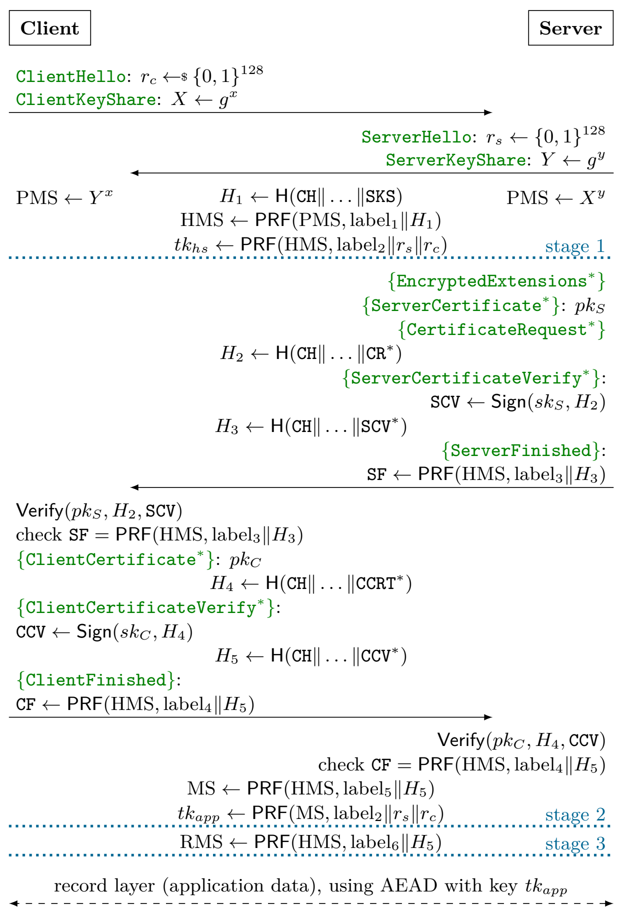
Client
Server
ClientHello: $r_c \leftarrow_{\$} \{0,1\}^{128}$
ClientKeyShare: $X \leftarrow g^x$
ServerHello: $r_s \leftarrow \{0,1\}^{128}$
ServerKeyShare: $Y \leftarrow g^y$
$\mathrm{PMS} \leftarrow Y^x$
$H_1 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SKS})$
$\mathrm{PMS} \leftarrow X^y$
$\mathrm{HMS} \leftarrow \mathsf{PRF}(\mathrm{PMS}, \mathrm{label}_1\|H_1)$
$tk_{hs} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_2\|r_s\|r_c)$
stage 1
{EncryptedExtensions*}
{ServerCertificate*}: pk_S
{CertificateRequest*}
$H_2 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CR}^*)$
{ServerCertificateVerify*}:
$\mathtt{SCV} \leftarrow \mathsf{Sign}(sk_S, H_2)$
$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SCV}^*)$
{ServerFinished}:
$\mathtt{SF} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_3\|H_3)$
$\mathsf{Verify}(pk_S, H_2, \mathtt{SCV})$
check $\mathtt{SF} = \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_3\|H_3)$
{ClientCertificate*}: pk_C
$H_4 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CCRT}^*)$
{ClientCertificateVerify*}:
$\mathtt{CCV} \leftarrow \mathsf{Sign}(sk_C, H_4)$
$H_5 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CCV}^*)$
{ClientFinished}:
$\mathtt{CF} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_4\|H_5)$
$\mathsf{Verify}(pk_C, H_4, \mathtt{CCV})$
check $\mathtt{CF} = \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_4\|H_5)$
$\mathrm{MS} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_5\|H_5)$
$tk_{app} \leftarrow \mathsf{PRF}(\mathrm{MS}, \mathrm{label}_2\|r_s\|r_c)$
stage 2
$\mathrm{RMS} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_6\|H_5)$
stage 3
record layer (application data), using AEAD with key tk_{app}

Figure 1: The handshake protocol in TLS 1.3 `draft-05`. `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.

parties authenticate 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 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.
- `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 g^{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}:

- `EncryptedExtensions` (`EE`) contains more extensions.

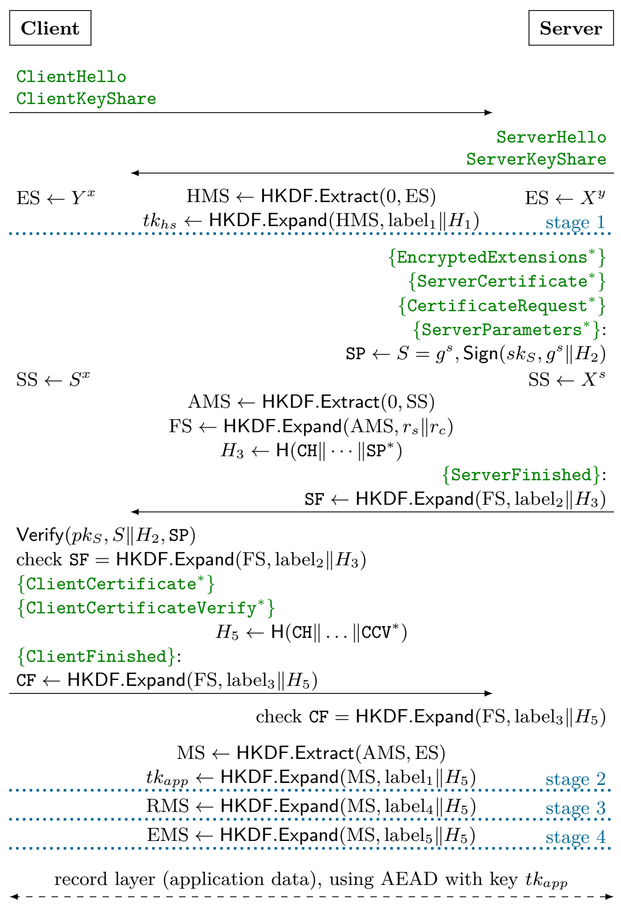
Client
Server
ClientHello
ClientKeyShare
ServerHello
ServerKeyShare
$\mathrm{ES} \leftarrow Y^x$
$\mathrm{HMS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{ES})$
$\mathrm{ES} \leftarrow X^y$
$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HMS}, \mathrm{label}_1\|H_1)$
stage 1
{EncryptedExtensions*}
{ServerCertificate*}
{CertificateRequest*}
{ServerParameters*}:
$\mathtt{SP} \leftarrow S = g^s, \mathsf{Sign}(sk_S, g^s\|H_2)$
$\mathrm{SS} \leftarrow S^x$
$\mathrm{SS} \leftarrow X^s$
$\mathrm{AMS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{SS})$
$\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{AMS}, r_s\|r_c)$
$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\|\cdots\|\mathtt{SP}^*)$
{ServerFinished}:
$\mathtt{SF} \leftarrow \mathsf{HKDF.Expand}(\mathrm{FS}, \mathrm{label}_2\|H_3)$
$\mathsf{Verify}(pk_S, S\|H_2, \mathtt{SP})$
check $\mathtt{SF} = \mathsf{HKDF.Expand}(\mathrm{FS}, \mathrm{label}_2\|H_3)$
{ClientCertificate*}
{ClientCertificateVerify*}
$H_5 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CCV}^*)$
{ClientFinished}:
$\mathtt{CF} \leftarrow \mathsf{HKDF.Expand}(\mathrm{FS}, \mathrm{label}_3\|H_5)$
check $\mathtt{CF} = \mathsf{HKDF.Expand}(\mathrm{FS}, \mathrm{label}_3\|H_5)$
$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{AMS}, \mathrm{ES})$
$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_1\|H_5)$
stage 2
$\mathrm{RMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_4\|H_5)$
stage 3
$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_5\|H_5)$
stage 4
record layer (application data), using AEAD with key tk_{app}

Figure 2: The handshake protocol in TLS 1.3 `draft-dh`. Hash value and message computations not stated explicitly are performed identically to TLS 1.3 `draft-05` (Figure 1). For unauthenticated handshakes, the `Server Certificate` and `ServerParameters` messages are omitted and key derivation proceeds with SS set to ES.

- `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`)/`ClientCertificate Verify` (`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.

2.2 draft-dh Handshake

Figure 2 shows the message flow and cryptographic computations 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. Key derivation is done using the HKDF extract-then-expand key derivation function [22], rather than the TLS PRF.

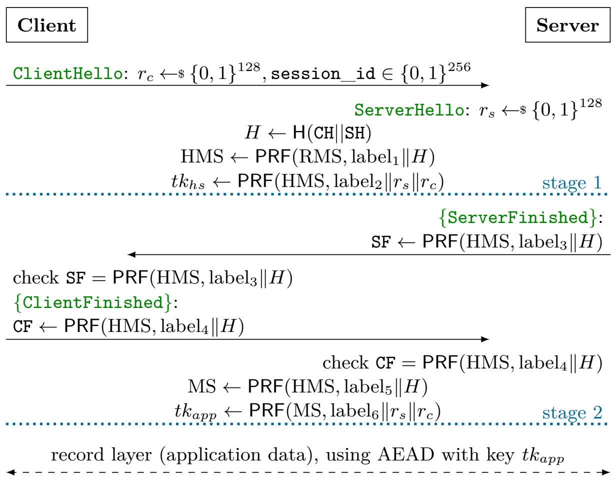
Client
Server
ClientHello: $r_c \leftarrow_\$ \{0,1\}^{128}$, session_id $\in \{0,1\}^{256}$
ServerHello: $r_s \leftarrow_\$ \{0,1\}^{128}$
$H \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$
$\mathrm{HMS} \leftarrow \mathsf{PRF}(\mathrm{RMS}, \mathrm{label}_1\|H)$
$tk_{hs} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_2\|r_s\|r_c)$
stage 1
{ServerFinished}:
$\mathtt{SF} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_3\|H)$
check $\mathtt{SF} = \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_3\|H)$
{ClientFinished}:
$\mathtt{CF} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_4\|H)$
check $\mathtt{CF} = \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_4\|H)$
$\mathrm{MS} \leftarrow \mathsf{PRF}(\mathrm{HMS}, \mathrm{label}_5\|H)$
$tk_{app} \leftarrow \mathsf{PRF}(\mathrm{MS}, \mathrm{label}_6\|r_s\|r_c)$
stage 2
record layer (application data), using AEAD with key tk_{app}

Figure 3: Session resumption in TLS 1.3 draft-05, using the resumption master secret RMS as a pre-shared key.

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 session_id of some previously established session, whose resumption master secret is used for the key derivation. We omit a description of resumption in draft-dh as its key schedule is not conclusively specified.

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.

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 [6], and necessitated non-standard assumptions in the security proof (e.g., the PRF-Oracle-Diffie–Hellman (PDF-ODH) assumption [19, 23]). 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.

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, 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 contrasts with the case of RSA key transport in the TLS 1.2 full handshake: the analyses of both Krawczyk et al. [23] and Bhargavan et al. [10] 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 [8] 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 session protection and authentication properties established by TLS in an application-layer protocol), including the session hash is appropriate. While the standardized tls-unique [4] and proposed tls-unique-prf [20] 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. In draft-dh session resumption, there is no ephemeral shared secret and the master secret is computed as a series of $\mathsf{HKDF.Extract}$ 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).

4. MULTI-STAGE KEY EXCHANGE

In this section we recap and extend the model for *multi-stage key exchange* by Fischlin and Günther [17] based on the Bellare–Rogaway-style model of Brzuska et al. [12, 11].

4.1 Outline

Our model for multi-stage key exchange protocols follows the Bellare–Rogaway paradigm. 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 [17] 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 [13]. 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 [17]).

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 [19], 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 $i \in \{0, \ldots, M\}$.[4]

- $\mathsf{label} \in \mathsf{LABELS}$: 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. $\mathsf{rejected}_i$
- $\mathsf{sid} \in (\{0,1\}^* \cup \{\bot\})^\mathsf{M}$: sid_i $[\bot]$ indicates the session identifier in stage $i > 0$
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\bot\})^\mathsf{M}$: cid_i $[\bot]$ indicates the contributive identifier in stage $i > 0$
- $\mathsf{K} \in (\{0,1\}^* \cup \{\bot\})^\mathsf{M}$: K_i $[\bot]$ indicates the established session key in stage $i > 0$
- $\mathsf{st}_\mathsf{key} \in \{\mathsf{fresh}, \mathsf{revealed}\}^\mathsf{M}$: $\mathsf{st}_{\mathsf{key},i}$ $[\mathsf{fresh}]$ indicates the state of the session key in stage $i > 0$
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^\mathsf{M}$: test indicator tested_i $[\mathsf{false}]$, 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., $\mathsf{label.sid}$ for the element sid in the tuple with label label in List_S, and analogously for other entries.

[4]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.

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 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 the authenticity for each stage. 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 simply speak of *no authentication* if all keys are unauthenticated and *stage-k unilateral (resp. mutual) authentication* if the keys of stages i are unauthenticated for $i < k$ and unilaterally (resp. mutually) authenticated for $i \geq k$.

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 [17] 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.

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 identifer kid for it.

$\mathsf{NewSession}(U, V, \mathsf{role}, \mathsf{auth}, \mathsf{kid}_U, \mathsf{kid}_V)$: 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 $\bot$. 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 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 $\bot$. 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[5] 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 $\bot$. 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.
In the non-forward-secret case, for each session label owned by U and all $i \in \{1, \ldots, \mathsf{M}\}$, set $\mathsf{label}.\mathsf{st}_{\mathsf{key},i}$ 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, $\mathsf{label}.\mathsf{st}_{\mathsf{key},i}$ 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 $\bot$. 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 List_S or if $\mathsf{label}.\mathsf{st}_\mathsf{exec} \neq \mathsf{accepted}_i$, return $\bot$. 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 $\mathsf{sid}_i = \mathsf{sid}'_i$, 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 $\mathsf{label}.\mathsf{auth}_i = \mathsf{unauth}$ or if $\mathsf{label}.\mathsf{auth}_i = \mathsf{unilateral}$ and $\mathsf{label}.\mathsf{role} = \mathsf{responder}$, 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 List_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.[6]
If $\mathsf{label}.\mathsf{tested}_i = \mathsf{true}$, return K_i, ensuring that repeated queries will be answered consistently.
Otherwise, set $\mathsf{label}.\mathsf{tested}_i$ to true. If the test bit b_test is 0, sample $\mathsf{label}.\mathsf{K}_i \leftarrow\$ \mathcal{D}$ 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 List_S with $\mathsf{sid}_i = \mathsf{sid}'_i$, also set $\mathsf{label}'.\mathsf{K}'_i \leftarrow \mathsf{label}.\mathsf{K}_i$ 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 $\mathsf{label}.\mathsf{K}_i$.

[5]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.

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

4.5 Security of Multi-Stage Key Exchange

The security properties for multi-stage key exchange protocols are split in two games, following Fischlin et al. [17] and Brzuska et al. [12, 11]. On the one hand, Match security ensures that the session identifiers sid effectively match the partnered sessions. On the other hand, $\mathsf{Multi\text{-}Stage}$ security ensures Bellare–Rogaway-like key secrecy.

Our notion of Match security—extended beyond [17] 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 concerning the session's derived keys, authentication type, contributive identifiers, and intended (authenticated) partner. Moreover, session identifiers must not match across different stages or be shared by more than two sessions.

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* $\mathsf{NewSession}$, Send, $\mathsf{NewTempKey}$, Reveal, *and* $\mathsf{Corrupt}$.

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* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$, $\mathsf{label.st_{exec}} \neq \mathsf{rejected}_i$, *and* $\mathsf{label}'.\mathsf{st_{exec}} \neq \mathsf{rejected}_i$, *but* $\mathsf{label.K}_i \neq \mathsf{label}'.\mathsf{K}_i$. *(Different session keys in some stage of partnered sessions.)*
2. *There exist two distinct labels* label, label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$, *but* $\mathsf{label.auth}_i \neq \mathsf{label}'.\mathsf{auth}_i$. *(Different authentication types in some stage of partnered sessions.)*
3. *There exist two distinct labels* label, label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$, *but* $\mathsf{label.cid}_i \neq \mathsf{label}'.\mathsf{cid}_i$ *or* $\mathsf{label.cid}_i = \mathsf{label}'.\mathsf{cid}_i = \bot$. *(Different or unset contributive identifiers in some stage of partnered sessions.)*
4. *There exist two distinct labels* label, label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$, $\mathsf{label.auth}_i = \mathsf{label}'.\mathsf{auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$, $\mathsf{label.role} = \mathsf{initiator}$, *and* $\mathsf{label}'.\mathsf{role} = \mathsf{responder}$, *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* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_j \neq \bot$ *for some stages* $i, j \in \{1, \dots, \mathsf{M}\}$ *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* $i \in \{1, \dots, \mathsf{M}\}$. *(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:* $\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1\right]$.

Definition 4.2 ($\mathsf{Multi\text{-}Stage}$ 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* $\mathsf{NewSession}$, Send, $\mathsf{NewTempKey}$, Reveal, $\mathsf{Corrupt}$, *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* $i \in \{1, \dots, \mathsf{M}\}$ *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* $\mathsf{Corrupt}$*) queries and the finalization step of the game; for example,* $\mathsf{Corrupt}$ *is defined differently for non-forward-secrecy versus stage-*j *forward secrecy.*

We say KE *is* $\mathsf{Multi\text{-}Stage}$*-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. DRAFT-05 HANDSHAKE SECURITY

We can now analyze the handshake as specified in TLS 1.3 `draft-05` [25].

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: $\mathsf{sid}_1 = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS})$, $\mathsf{sid}_2 = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}^*, \mathtt{SCRT}^*, \mathtt{CR}^*, \mathtt{SCV}^*, \mathtt{CCRT}^*, \mathtt{CCV}^*)$. 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 $\mathsf{Multi\text{-}Stage}$ 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 $\mathsf{cid}_1 = (\mathtt{CH}, \mathtt{CKS})$ and subsequently, on receiving (resp. sending) the `ServerHello` and `ServerKeyShare` messages, extend it to $\mathsf{cid}_1 = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS})$. 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 bitlength of the nonces.*

Match security follows from the way the session identifiers are chosen (to include all unencrypted messages), in particular guarantees that partnered sessions derive the same key, authenticity, and contributive identifiers. The given security bound takes into account the probability that three honest session chose the same nonce and group element. The proof appears in the full version [15].

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 concurrently providing 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* $\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-05},\mathcal{A}} \leq 3n_s \cdot \big[\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_3} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_4} + n_s \cdot \big(\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_5} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{PRF},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{B}_8}\big)\big]$*, 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 sketch. We only sketch the main steps of the proof in this version here due to space restrictions; see the full version [15]. The proof uses the common game-hopping techniques to bound the adversary's success probability.

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.[7] The hybrid details are omitted here due to space restrictions; see the full version for details [15].

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.

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

1. the adversary tests a client session without honest contributive partner in the first stage,
2. the adversary tests a server session without honest contributive partner in the first stage, and
3. the tested session has an honest contributive partner in stage 1.

Case A. Test client without partner. First we abort if any two sessions compute the same value for different hash inputs, and use a hash-collision challenger with algorithm $\mathcal{B}_1$ to bound the difference in advantage. Without honest partner, the client session must have an authenticated peer in order to allow testing and hence needs to receive a signature that, however, no honest server will sent (otherwise, algorithm $\mathcal{B}_2$ will have a signature forgery). This leads to a situation where no client accepts without a partner.

Case B. Test server without partner. This case proceeds virtually identically to the previous case (in particular with the same game hops and associated probabilities for hash-collision algorithm $\mathcal{B}_3$ and signature forger $\mathcal{B}_4$), this time assuring partnering due to the `CCV` message sent.

Case C. Test with partner. We proceed as follows.

Game C.1. We first guess the contributively partnered session label label′ and abort $\mathsf{label.cid}_1 \neq \mathsf{label'.cid}_1$, bounding the probability of an abort even by n_s.

Game C.2. In this game we replace the premaster secret PMS in the tested and partner session with random value $\widetilde{\mathrm{PMS}} = g^z$ (where $z \leftarrow_\$ \mathbb{Z}_q$), using a DDH challenger to replace g^x, g^y, PMS with g^u, g^v, h bounding the difference in the advantage of $\mathcal{A}$ between Games C.1 and C.2 by the probability of algorithm $\mathcal{B}_5$ breaking the DDH assumption.

Game C.3. In this game we replace the handshake master secret PMS in the tested and partner session with random value $\widetilde{\mathrm{HMS}} \leftarrow_\$ \{0,1\}^\lambda$ using a PRF challenger with $\widetilde{\mathrm{PMS}}$ as the PRF key, bounding the difference in the advantage of $\mathcal{A}$ between Games C.2 and C.3 by the probability of success of algorithm $\mathcal{B}_6$ breaking the PRF assumption.

Game C.4. In this game we replace the handshake traffic key tk_{hs}, the resumption master secret RMS and the master secret MS with uniformly random values $\widetilde{tk_{hs}}$, $\widetilde{\mathrm{MS}}$, $\widetilde{\mathrm{RMS}} \leftarrow_\$ \{0,1\}^\lambda$ respectively (the output from a PRF challenger using $\widetilde{\mathrm{HMS}}$ as a PRF key), bounding the difference in advantage for $\mathcal{A}$ between Games C.3 and C.4 with the success of algorithm $\mathcal{B}_7$ in breaking the PRF assumption.

Game C.5. In this game we replace the application traffic key tk_{app} with uniformly random values $\widetilde{tk_{app}} \leftarrow_\$ \{0,1\}^\lambda$ respectively (the output from a PRF challenger using $\widetilde{\mathrm{MS}}$ as a PRF key), bounding the difference in advantage for $\mathcal{A}$ between Games C.3 and C.4 with the success of algorithm $\mathcal{B}_8$ in breaking the PRF assumption.

Note that in game C.5 the session keys $\widetilde{tk_{hs}}$ and $\widetilde{tk_{app}}$ as well as the resumption master secret $\widetilde{\mathrm{RMS}}$ are now chosen independently from the protocol run and uniformly at random. As the response to the Test query is now independent of the test bit b_{test}, $\mathcal{A}$ cannot distinguish whether it is given the real or random key, and combining the various bounds yields the above security bound. □

6. DRAFT-DH HANDSHAKE SECURITY

We now analyze the TLS 1.3 handshake variant as specified in the `draft-dh` fork by Rescorla [27]. Session and contributive identifiers are defined and set as for `draft-05` (in particular again over the *unencrypted* messages, for key independence), except for renaming the `ServerCertificate Verify` message to `ServerParameters` and adding a fourth stage for the exporter master secret EMS derived with a unique sid label "EMS". The semi-static keys g^s involved in `draft-dh` on the server side are captured as temporary

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

keys in our model, allowing the adversary to decide which value to use in each session. Exposure of the server's semi-static keys s is not considered in our analysis (in particular they are not revealed by Corrupt). We stress that, while our Multi-Stage security result below would indeed hold even under full exposure of of all values s in use (due to its non–security-critical influence on the full handshake's key derivation), the envisioned 0-RTT keys will critically rely on the contributing s being secret. Hence, acceptable conditions for its exposure need to be carefully crafted once the 0-RTT handshake is fully specified.

Theorem 6.1 (Match security of `draft-dh`). *The* `draft-dh` *full handshake is* Match*-secure: for any efficient adversary* $\mathcal{A}$ *we have* $\mathsf{Adv}^{\mathsf{Match}}_{\mathtt{draft\text{-}dh},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|}$*, where* n_s *is the maximum number of sessions,* q *is the group order, and* $|nonce| = 128$ *is the bitlength 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 `Server Parameters` 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 concurrently providing 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* $\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{draft\text{-}dh},\mathcal{A}} \leq 4n_s \cdot \Big[\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_3} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_4} + n_s \cdot \Big(\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{HKDF.Extract},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_9} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}}\Big)\Big]$*, where* n_s *is the maximum number of sessions and* n_u *is the maximum number of users.*

Proof sketch. We again only sketch the main points and again refer to the full version for details [15].

First, as in the `draft-05` proof we consider a single Test query only (reducing the advantage by a factor of $4n_s$ due to four stages by the same hybrid argument) and split the analysis in the same three cases as for `draft-05`. Cases A and B are virtually identical to the respective cases in the `draft-05` proof, aside from renaming the server's signature message to `ServerParameters`.

In the proof steps of the third case, we first guess the contributively partnered session (introducing a factor of n_s) and abort in case of a hash collision (bounded by the collision resistance of H) or signature forgery on g^s and the session hash in `ServerParameters` (bounded by the signature's EUF-CMA security). We are then ensured that the challenger controls all key inputs in the tested session (as well as its potential partner) and can hence gradually replace the keys derived within the following steps. First, we replace ES with a random value (bounded by winning a DDH challenge encoded in g^x, g^y, and ES). Second, we choose HMS and MS at random (bounded by the security of HKDF.Extract modelled as pseudorandom function using a now uniformly random ES). Finally, we replace the HKDF.Expand evaluations under HMS and MS, which are now uniformly random, by random functions (bounded twice by the security of HKDF.Expand, also modelled as pseudorandom function). The Test response is now independent of b_{test}). □

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. [12] for Bellare–Rogaway-secure key exchange protocols and by Fischlin and Günther [17] 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 [17] 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.

As established by Brzuska et al. [12], 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 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.

Due to space limitations, we can only provide a summary of our composition result here; details appear in the full version [15]. We first extend the syntax of composed games introduced by Brzuska et al. [12, 11] and extended by Fischlin and Günther [17] 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. Moreover, we strengthen their notion of session matching to capture *non-public partnering*.

We can then 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 $\mathsf{Multi\text{-}Stage}$-secure providing *key independence*, *stage-j forward secrecy* (for $j \leq i$), and *multi-stage session matching*. Observe that we, in contrast to the previous composition result [17], do *not* require a particular level of authentication, but instead show compositional security for any concurrent authentication properties AUTH of KE. We remark 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 7.1 (Multi-stage composition)**.** *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 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 $i \geq j$ 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 $\mathsf{Adv}^{G_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{B}_1} + n_s \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}_2} + \mathsf{Adv}^{G_\Pi}_{\Pi,\mathcal{B}_3}$ where n_s is the maximum number of sessions in the key exchange game.*

The proof is omitted here due to space restrictions; it follows the one for multi-stage composition [17], and appears in the full version [15].

8. PRESHARED-SECRET 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. 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 `Client Hello` 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.
- $\mathsf{Corrupt}(U, V, k)$: 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

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.

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. The Match security conditions are identical for MS-PSKE and MSKE with the following exception (condition 4):

4. sessions are partnered with the intended (authenticated) participant, and for mutual authentication share the same key index.

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 conditions from Definition 4.1 (with the following condition replacing condition 4) hold:

4. *There exist two distinct labels* label*,* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$*,* $\mathsf{label.auth}_i = \mathsf{label'.auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$*,* label.role = initiator*, and* label′.role = responder*, but* $\mathsf{label}.V \neq \mathsf{label'}.U$*, or* $\mathsf{label}.k \neq \mathsf{label'}.k$*, or, when* $\mathsf{label.auth}_i = \mathsf{mutual}$*,* $\mathsf{label}.U \neq \mathsf{label'}.V$*.*

We say KE *is* Match*-secure if for all adversaries $\mathcal{A}$ the following advantage is negligible in the security parameter:* $\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1\right]$.

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* lost ← 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* lost ← 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* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i$*,* $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$*, and* $\mathsf{label'.tested}_i = \mathsf{true}$*. (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 lost = 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. DRAFT-05 RESUMPTION SECURITY

We now turn towards session resumption and analyze the resumption handshake as specified in the `draft-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`.

Let $\mathsf{sid}_1 = \mathsf{cid}_1 = (\texttt{ClientHello}, \texttt{ServerHello}, \text{“stage1”})$ 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 “*servername* ‖ *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 preshared 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 sketch. Match security follows from the session identifiers including the nonces and the identifier psid of the preshared secret, where the latter also maps to the intended partner. Hence identical session identifiers lead to agreement on keys and communication partner, while the random nonces used can be used to bound the probability of three honest sessions choosing the same nonce by $n_s^2 \cdot 2^{-|nonce|}$. □

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

Proof sketch. As in the main `draft-05` Multi-Stage proof (Theorem 5.2), considering a single Test query introduces a factor of $1/2n_s$. We then proceed by first ruling out collisions under H (due to its collision resistance) before we gradually replace evaluations of PRF by random functions, bounded three times by the PRF security: Firstly, in all sessions sharing the tested preshared secret $\mathsf{pss}_{U,V,k}$ (including the partner session), we replace the handshake master secret value HMS with a random value (leveraging that the sent psid uniquely identifies the secret which must be uncorrupted and hence uniformly random for $\mathcal{A}$). We then replace evaluations under, secondly, HMS and, thirdly, MS resulting in tk_{hs} and tk_{app} to be chosen uniformly at random and the Test response being independent of b_{test}. □

10. CONCLUSION

Our successful analysis of both `draft-05` and `draft-dh` is encouraging: both TLS 1.3 candidate handshake designs can be shown to be cryptographically sound, even when restricting to standard cryptographic assumptions only. The analysis also reveals 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.

11. REFERENCES

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *ACM CCS 15*, 2015.

[2] N. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the security of RC4 in TLS. In *Proc. 22nd USENIX Security Symposium*, pages 305–320, 2013.

[3] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013.

[4] J. Altman, N. Williams, and L. Zhu. Channel Bindings for TLS. RFC 5929 (Proposed Standard), 2010.

[5] C. Badertscher, C. Matt, U. Maurer, P. Rogaway, and B. 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`.

[6] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO'93*, pages 232–249, 1994.

[7] B. Beurdouche, K. Bhargavan, A. Delignat-Levaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. 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, 2015.

[8] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P.-Y. Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113, 2014.

[9] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub. Implementing TLS with verified cryptographic security. In *2013 IEEE Symposium on Security and Privacy*, pages 445–459, 2013.

[10] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella Béguelin. Proving the TLS handshake secure (as it is). In *CRYPTO 2014, Part II*, pages 235–255, 2014.

[11] C. Brzuska. *On the Foundations of Key Exchange.* PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2013. `http://tuprints.ulb.tu-darmstadt.de/3414/`.

[12] C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams. Composability of Bellare-Rogaway key exchange protocols. In *ACM CCS 11*, pages 51–62, 2011.

[13] R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *CRYPTO 2002*, pages 143–161, 2002. `http://eprint.iacr.org/2002/120/`.

[14] Codenomicon. The Heartbleed bug. `http://heartbleed.com`, 2014.

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

[16] T. Duong. BEAST. `http://vnhacker.blogspot.com.au/2011/09/beast.html`, 2011.

[17] M. Fischlin and F. Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In *ACM CCS 14*, pages 1193–1204, 2014.

[18] C. Fournet, M. Kohlweiss, and P.-Y. Strub. Modular code-based cryptographic verification. In *ACM CCS 11*, pages 341–350, 2011.

[19] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO 2012*, pages 273–293, 2012.

[20] S. Josefsson. Channel bindings for TLS based on the PRF. `https://tools.ietf.org/html/draft-josefsson-sasl-tls-cb-03`, 2015.

[21] M. Kohlweiss, U. Maurer, C. Onete, B. Tackmann, and D. Venturi. (de-)constructing TLS. Cryptology ePrint Archive, Report 2014/020, 2014. `http://eprint.iacr.org/2014/020`.

[22] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *CRYPTO 2010*, pages 631–648, 2010.

[23] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In *CRYPTO 2013, Part I*, pages 429–448, 2013.

[24] B. Möller, T. Duong, and K. Kotowicz. This POODLE bites: Exploiting the SSL 3.0 fallback. `https://www.openssl.org/~bodo/ssl-poodle.pdf`, 2014.

[25] 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`, 2015.

[26] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-07. `https://tools.ietf.org/html/draft-ietf-tls-tls13-07`, 2015.

[27] 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`, 2015.