J Cryptol (2021)34:37
https://doi.org/10.1007/s00145-021-09384-1

Journal of CRYPTOLOGY

A Cryptographic Analysis of the TLS 1.3 Handshake Protocol

Benjamin Dowling
Department of Computer Science, ETH Zürich, Zurich, Switzerland

Marc Fischlin
TU Darmstadt, Darmstadt, Germany
marc.fischlin@cryptoplexity.de

Felix Günther
Department of Computer Science, ETH Zürich, Zurich, Switzerland

Douglas Stebila
University of Waterloo, Waterloo, Canada
dstebila@uwaterloo.ca

Communicated by Colin Boyd

Received 31 October 2019 / Revised 22 February 2021 / Accepted 22 February 2021
Online publication 30 July 2021

Abstract. We analyze the handshake protocol of the Transport Layer Security (TLS) protocol, version 1.3. We address both the full TLS 1.3 handshake (the one round-trip time mode, with signatures for authentication and (elliptic curve) Diffie–Hellman ephemeral ((EC)DHE) key exchange), and the abbreviated resumption/"PSK" mode which uses a pre-shared key for authentication (with optional (EC)DHE key exchange and zero round-trip time key establishment). Our analysis in the reductionist security framework uses a multi-stage key exchange security model, where each of the many session keys derived in a single TLS 1.3 handshake is tagged with various properties (such as unauthenticated versus unilaterally authenticated versus mutually authenticated, whether it is intended to provide forward security, how it is used in the protocol, and whether the key is protected against replay attacks). We show that these TLS 1.3 handshake protocol modes establish session keys with their desired security properties under standard cryptographic assumptions.

Keywords. Authenticated key exchange, Transport Layer Security (TLS), Handshake protocol.

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. Originally developed as the Secure Sockets Layer (SSL) protocol version 3 in 1996, TLS version 1.0 was standardized by the Internet Engineering Task Force (IETF) in 1998 [37], with subsequent revisions to version 1.1 (2006) [48] and version 1.2 (2008) [49]. Despite its large-scale deployment, or perhaps because of it, we have witnessed frequent successful attacks against TLS. Starting around 2009, there were many practical attacks on the then-current version 1.2 of TLS that received significant attention, exploiting weaknesses in underlying cryptographic primitives (such as weaknesses in RC4 [4]), errors in the design of the TLS protocol (e.g., BEAST [51], the Lucky 13 attack [7], the triple handshake attack [13], the POODLE attack [84], the Logjam attack [2]), or flaws in implementations (e.g., the Heartbleed attack [35], state machine attacks (SMACK [10])).

1.1. *Development and Standardization of TLS 1.3*

With concerns rising about the security of TLS version 1.2 due to the many attacks, but also motivated by desire to deprecate old algorithms, enhance privacy, and reduce connection establishment latency, in 2014 the IETF's TLS working group initiated a multi-year process to develop and standardize a new version of TLS, eventually called version 1.3. From 2014 through 2018, a total 29 drafts of TLS 1.3 were published, with active feedback from industry and academia, including extensive security analyses by various teams from academia (see [89] for a chronicle of the development and analysis of TLS 1.3). The document standardizing TLS 1.3, RFC 8446 [90], was published in August 2018 and has already seen widespread adoption.

From a cryptographic perspective, major design changes in TLS 1.3 compared to version 1.2 include: (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) using different keys to encrypt handshake messages and 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 protecting application data; and (7) providing handshakes with fewer message flows to reduce latency.

There are two primary modes of the TLS 1.3 handshake protocol. One is the full, one round-trip time (1-RTT) handshake, which uses public-key certificates for server and (optionally) client authentication, and (elliptic curve) Diffie–Hellman ephemeral ((EC)DHE) key exchange, inspired by Krawczyk's 'SIGn-and-MAc' (SIGMA) design [72]. Several session keys are established for a variety of purposes in this mode: to encrypt part of the handshake, to enable export of keying material to other applications, for session resumption, and of course to encrypt application data. This mode gets its name from the fact that application data can be sent from the client to the server with the handshake's completion after a full round trip, meaning there is one round-trip time (1-RTT) until the first application message can be sent (not counting non-TLS networking operations such as DNS lookups or the TCP 3-way handshake).

The other primary mode of the TLS 1.3 handshake protocol is the resumption or pre-shared key (PSK) mode, in which authentication is based on a symmetric pre-shared key, with optional (EC)DHE key exchange for forward secrecy; this generalizes the abbreviated session resumption handshake from earlier versions of TLS. The PSK mode can optionally be augmented with a zero round-trip time (0-RTT) key establishment, allowing the client to send—along with its first TLS flow—application data encrypted under a key derived from the PSK.

1.2. *Security Analyses of TLS*

TLS 1.2 and Prior Versions A long line of work has analyzed various versions of the SSL/TLS protocol using both formal methods and reductionist security proofs. In the reductionist security paradigm, early work [56,63,86] on the handshake protocol dealt with modified or truncated versions of the protocol, necessary because TLS 1.2 and earlier did not have strict key separation: the session key was also used to encrypt messages within the handshake protocol, barring security proofs in strong indistinguishability-based authenticated key exchange models in the Bellare–Rogaway [25] style. There were also formalizations of the security of the authenticated encryption in the record layer [71,87]. A major milestone in reductionist analyses of TLS was the development of the authenticated and confidential channel establishment (ACCE) security model which allowed for the combined analysis of a full TLS 1.2 handshake and secure channel in a single model [64], sidestepping the aforementioned key separation issue; this work was followed by a range of other works analyzing the security of various aspects of TLS 1.2 [50,58,70,76,81]. Other approaches to proving the security of TLS 1.2 within the reductionist security paradigm include a range of modular and compositional approaches [21] as well as approaches that combine formal analysis and reductionist security [18,19].

TLS 1.3 Drafts The handshake protocol in initial drafts of TLS 1.3 was based in part on the OPTLS protocol [77]. There were a variety of investigations on the security of various drafts throughout the TLS 1.3 standardization process. Using the reductionist security paradigm, there have been analyses of the handshake protocol [15,38,39,53,55,69,75, 77,82] and the record layer [24,27,60,80,88]. There has been a range of work involving formal methods and tools, such as model checkers and symbolic analysis [29,31], and approaches combining verified implementations with formal analysis and reductionist security [9,11,20,40].

TLS 1.3 Standard Since TLS 1.3 was published as an RFC in August 2018, some works have addressed the final TLS 1.3 standard. The Selfie attack [44] led to updated analyses of PSK handshakes [1,44]. Arfaoui et al. [3] investigated the privacy features of the TLS 1.3 handshake. Revised computational security proofs of the full 1-RTT handshake by Diemert and Jager [45] and Davis and Günther [43] translated techniques of Cohn-Gordon et al. [28] to establish tighter reductions. There have also been academic proposals for improvements to or modifications of TLS 1.3, considering forward security for the 0-RTT handshake [6], running TLS 1.3 over a different network protocol [32], or defining a KEM-based alternative handshake enabling the deployment of post-quantum schemes [93].

1.3. *Our Contributions*

We give a reductionist security analysis of three modes of the TLS 1.3 handshake: the full 1-RTT handshake, the PSK handshake (with optional 0-RTT mode), and the PSK-(EC)DHE handshake (with optional 0-RTT mode); based on a cryptographic abstraction of the protocols we provide in Sect. 3. In order to carry out our analysis, we formalize a multi-stage key exchange security model which can capture a variety of characteristics associated to each stage key. Our analysis shows that the design of the TLS 1.3 handshake follows sound cryptographic principles.

Security Model Our security model, given in Sect. 4, follows the Bellare–Rogaway (BR) model [25] for authenticated key exchange security based on session key indistinguishability, as formalized by Brzuska et al. [22,26], and our model builds specifically on the multi-stage model of Fischlin and Günther [52,61]. The latter deals with key exchange protocols that derive a series of session keys in the course of multiple protocol stages. Our extension of their multi-stage key exchange model allows us to capture the following characteristics associated to the session key established at each stage, which we call the stage key:

- *Authentication*: whether a stage key is unauthenticated, unilaterally authenticated, or mutually authenticated. We further extend the multi-stage model to capture *upgradable authentication*: a stage's key may be considered, say, unauthenticated at the time it is accepted, but the authentication level of this key may be "raised" to unilaterally authenticated or, potentially in a second step, mutually authenticated after some later operations, such as verification of a signature in a later message.
- *Forward secrecy*: whether a stage key is meant to provide forward secrecy, namely that it remains secure after compromise of a long-term secret involved in its derivation.
- *Key usage*: whether a stage key is meant to be used internally within the protocol (for example, to encrypt later handshake messages), or externally (for example, composed with a symmetric encryption scheme to protect application messages or used in some other external symmetric-key protocol).
- *Replayability*: whether it is guaranteed that a stage key is not established in result of a replay attack; early stages of the 0-RTT modes do not have this guarantee.

Our security model comes in two flavors that capture security established through two types of credentials: public keys or symmetric pre-shared keys. Following the BR model, our model of compromise includes long-term key compromise ($\mathsf{Corrupt}$) and stage key compromise (Reveal). While other models [33,79] further capture the compromise of session state or ephemeral randomness, TLS is not designed to be secure against such exposure of ephemeral values and we hence do not include these compromise capabilities in our model.

In addition to capturing indistinguishability of stage keys, the model also ensures soundness of session identifiers using the Match-security notion of [22,26].

Protocol Analysis We apply our multi-stage key exchange security model in Sects. 5 and 6 to analyze the three modes of the TLS 1.3 handshake: full 1-RTT, PSK, and PSK-(EC)DHE, with the latter two having optional 0-RTT keys. There are four main classes of stage keys covered in the analysis: early data encryption and export keys (ETS, EEMS, only present in the PSK with 0-RTT modes); handshake traffic secrets ($\mathrm{tk}_{\mathrm{chs}}, \mathrm{tk}_{\mathrm{shs}}$); application traffic secrets (CATS, SATS); and exported keys (RMS for session resumption, EMS for other exported keys). This results in six stage keys in the full 1-RTT mode and eight stage keys in the PSK modes.

As noted above, our security model allows us to precisely capture various characteristics of different stage keys. For example, consider the client handshake traffic secret $\mathrm{tk}_{\mathrm{chs}}$, used to encrypt handshake messages from the client to the server. In the full 1-RTT handshake, this key is initially unauthenticated, then unilaterally authenticated through a server signature after stage 3 is reached, and may ultimately be mutually authenticated after stage 6 is reached if the client authenticates; it is forward secret; it is intended for internal use within the protocol; and it is guaranteed to be non-replayed. In contrast, in the PSK handshake, this key is mutually authenticated as soon as it is established, but does not have forward secrecy. Finally, in the PSK-EC(DHE) handshake, this key is unauthenticated initially, then is upgraded to unilateral and eventually mutual authentication after stages 5 and 8, when MACs within the `Finished` messages are verified; and it is forward secret.

The reductions showing the security of the protocol modes in the model follow a game hopping technique, and mainly rely on standard signature resp. MAC scheme unforgeability (for authentication in the full 1-RTT resp. PSK handshake), hash function collision resistance, PRF security (and in some cases dual PRF security), and an interactive Diffie–Hellman assumption (a variant of the PRF-Oracle-Diffie–Hellman assumption [17,64] called $\mathsf{dual\text{-}snPRF\text{-}ODH}$).

Observations on the Design and Security of TLS 1.3 In Sect. 7, we include a discussion about various characteristics of TLS 1.3 based on results of our security analysis, including how a variety of TLS 1.3 design decisions positively impact the security analysis (key separation and key independence, including the session hash in signatures and key derivation), some subtleties on the role of handshake encryption and key confirmation via `Finished` messages, as well as the susceptibility of 0-RTT keys to replays.

Relation to Our Earlier Work This paper is successor work to [38,39] and [53], as well as [47,61]. In [38], we first extended the multi-stage key exchange model of [52] as needed, then applied it to analyze two early drafts of TLS 1.3: `draft-05`, which has the same basic signed-Diffie–Hellman structure but a simplified key schedule compared to the final version, and an alternative proposal called `draft-dh` incorporating ideas from the OPTLS design [77], in which servers could have a semi-static DH key share. In [39], we updated our analysis to `draft-10` and added an analysis of the, by then revised, pre-shared-key handshake mode. In [53], a subset of us analyzed the 0-RTT pre-shared key and PSK-(EC)DHE mode in `draft-14`, as well as the later deprecated Diffie–Hellman-based 0-RTT mode using semi-static DH key shares in `draft-12`, which introduced the notion of replayable stages into the multi-stage key exchange security model. In a PhD thesis [47], one of us updated the work from [39] to address the full, PSK, and PSK-

(EC)DHE handshakes in `draft-16`; in another PhD thesis [61], another of us unified the MSKE model and the aforementioned results on the full and PSK handshakes of `draft-10` and the 0-RTT handshakes of `draft-12` and `draft-14`.

This paper updates this prior work to the final version of TLS 1.3 as published in RFC 8446 [90] (recall that there were 29 drafts leading up to the final standard). It addresses, in a unified security model, the full, PSK, and PSK-(EC)DHE handshakes, the latter two with optional 0-RTT keys. The security model in this paper includes enhancements not present in earlier works, particularly for capturing upgradable authentication. The model and analysis for the PSK mode have been updated to reflect the observations of Drucker and Gueron's "Selfie" attack [44] by associating intended roles with a pre-shared key.

Section 7.1 provides more details on technical differences between this paper and our earlier work.

Limitations The TLS 1.3 protocol allows users to support and negotiate different cryptographic algorithms including the used signature schemes, Diffie–Hellman groups, and authenticated encryption schemes. Many implementations will simultaneously support TLS 1.3, TLS 1.2, and even earlier versions. We do not aim to capture the security of this negotiation process nor security when a cryptographic key (e.g., a signing key) is re-used across different algorithm combinations or with earlier versions of TLS [66]. For the PSK modes of TLS 1.3, we do not treat how parties negotiate which pre-shared key to use. Our analysis assumes that all parties use only TLS 1.3 with a single combination of cryptographic algorithms and do not re-use keying material outside of that context (beyond consuming session keys established by the TLS 1.3 handshake).

In our proofs of key indistinguishability for all three TLS 1.3 handshake modes, some of our proof steps involve guessing parties and/or sessions, and thus are non-tight, similar to most proofs of authenticated key exchange protocols. Recently, Diemert and Jager [45] as well as Davis and Günther [43] have established new security proofs for the TLS 1.3 full 1-RTT handshake with tight reductions to the strong Diffie–Hellman assumption, translating techniques of Cohn-Gordon et al. [28].

Our focus is entirely on the TLS 1.3 handshake protocol, and thus does not address security of the record layer's authenticated encryption. TLS 1.3 also includes a variety of additional functionalities outside the core handshake that we treat as out of scope. Examples include session tickets, post-handshake authentication [75], the alert protocol, and changes for Datagram TLS (DTLS) 1.3 [92], as well as other extensions to TLS 1.3 currently in the Internet-Draft state.

Security in practice obviously relies on many more factors, such as good implementations and good operational security, which are important but outside the scope of this analysis.

2. Preliminaries

We begin with introducing the basic notation we use in this paper and recapping some core building blocks and cryptographic assumptions employed in our security analysis.

2.1. *Notation*

With $\mathbb{N}$ we denote the natural numbers. We write a bit as $b \in \{0, 1\}$ and a (bit) string as $s \in \{0, 1\}^*$, with $|s|$ indicating its (binary) length; $\{0, 1\}^n$ is the set of bit strings of length n. We write $x \leftarrow y$ for the assignment of value y to the variable x and $x \leftarrow_\$ X$ for uniformly sampling x from a (finite) set X.

For an algorithm $\mathcal{A}$ we write $x \leftarrow \mathcal{A}(y)$, resp. $x \leftarrow_\$ \mathcal{A}(y)$, for the algorithm deterministically, resp. probabilistically, outputting x on input y. We indicate by $\mathcal{A}^{\mathcal{O}}$ an algorithm $\mathcal{A}$ running with oracle access to some other algorithm $\mathcal{O}$.

2.2. *Collision-Resistant Hash Functions*

As often the case in practice, the cryptographic hash functions used in TLS 1.3 are unkeyed. When considering a hash function's collision resistance, we hence demand that a security reduction provides effective means for constructing a concrete algorithm generating a collision (cf. Rogaway [91]).

Definition 2.1. (*Hash function and collision resistance*) A *hash function* $\mathsf{H}\colon \{0, 1\}^* \to \{0, 1\}^\lambda$ maps arbitrary-length messages $m \in \{0, 1\}^*$ to a hash value $\mathsf{H}(m) \in \{0, 1\}^\lambda$ of fixed length $\lambda \in \mathbb{N}$.

We can now measure the collision resistance (COLL) with respect to an adversary $\mathcal{A}$ via the advantage

$$\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{A}} := \Pr\left[(m, m') \leftarrow_\$ \mathcal{A} : m \neq m' \text{ and } \mathsf{H}(m) = \mathsf{H}(m')\right].$$

In the common asymptotic notion, we would demand that one cannot construct an efficient adversary $\mathcal{A}$ where this advantage is non-negligible with respect to the security parameter λ.

2.3. *HMAC and HKDF*

TLS 1.3 employs HKDF [68,73] as its key derivation function, with HMAC [12,67] at its core. We briefly recap their definition and usage.

HMAC [12,67] is based on a cryptographic hash function $\mathsf{H}\colon \{0, 1\}^* \to \{0, 1\}^\lambda$ and keyed with some key $K \in \{0, 1\}^\lambda$ (larger key material is hashed through H to obtain a λ-bit key). Computing the HMAC value on some message m is then defined as $\mathsf{HMAC}(K, m) := \mathsf{H}((K \oplus \mathsf{opad}) \parallel \mathsf{H}((K \oplus \mathsf{ipad}) \parallel m))$, where opad and ipad are two λ-bit padding values consisting of repeated bytes `0x5c` and `0x36`, respectively.

HKDF follows the *extract-then-expand* paradigm for key derivation [68,73], instantiated with HMAC. We adopt the standard notation for the two HKDF functions: $\mathsf{HKDF.Extract}(\mathit{XTS}, \mathit{SKM})$ on input an (non-secret and potentially fixed) extractor salt XTS and some (not necessarily uniform) source key material SKM outputs a pseudorandom key PRK. $\mathsf{HKDF.Expand}(\mathit{PRK}, \mathit{CTXinfo}, L)$ on input a pseudorandom key PRK (from the $\mathsf{Extract}$ step) and some (potentially empty) context information $\mathit{CTXinfo}$ outputs pseudorandom key material KM of length L bits. (For simplicity, we omit the third parameter L in Expand when $L = \lambda$, which is the case throughout TLS 1.3 except when deriving traffic keys (cf. Table 2).) Both functions are instantiated with HMAC, where directly $\mathsf{HKDF.Extract}(XTS, SKM) := \mathsf{HMAC}(XTS, SKM)$ and $\mathsf{HKDF.Expand}$ iteratively invokes HMAC to generate pseudorandom output of the required length (see [73]).

2.4. *Dual PRF Security and the PRF-ODH Assumption*

Most key derivation steps in TLS 1.3 rely on regular pseudorandom function (PRF) security for the HKDF and HMAC functions. In our analysis of the PSK handshakes, we also treat HMAC as a collision-resistant unkeyed hash function over the pair of inputs, as in Definition 2.1. For some of its applications, we, however, need to deploy stronger assumptions which we recap here.

The first assumption is concerned with the use of HMAC as a dual PRF (cf. [14]).

Definition 2.2. (*Dual PRF security*) Let $f : \mathcal{K} \times \mathcal{L} \to \mathcal{O}$ be a pseudorandom function with key space $\mathcal{K}$ and label space $\mathcal{L}$ such that $\mathcal{K} = \mathcal{L}$. We define the *dual PRF security* of f as the PRF security of $f^{\mathsf{swap}}(k, l) := f(l, k)$ and the according advantage function as

$$\mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{f,\mathcal{A}} := \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{f^{\mathsf{swap}},\mathcal{A}}.$$

The second assumption, the so-called pseudorandom-function oracle-Diffie–Hellman $\mathsf{PRF\text{-}ODH}$ assumption, has been introduced by Jager et al. [64] in their analysis of the TLS 1.2 key exchange. It is a variant of the oracle-Diffie–Hellman assumption introduced by Abdalla et al. [5] in the context of the DHIES encryption scheme. Basically, the $\mathsf{PRF\text{-}ODH}$ assumption states that the value $\mathsf{PRF}(g^{uv}, x^\star)$ for a Diffie–Hellman-type key g^{uv} is indistinguishable from a random string, even when given g^u and g^v and when being able to see related values $\mathsf{PRF}(S^u, x)$ and/or $\mathsf{PRF}(T^v, x)$ for chosen values S, T, and x. The $\mathsf{PRF\text{-}ODH}$ assumption comes in various variants, which have been generalized and studied by Brendel et al. [17].

For our analysis of TLS 1.3, we will deploy only the $\mathsf{snPRF\text{-}ODH}$ assumption providing limited oracle access to only a single related value $\mathsf{PRF}(S^u, x)$, as well as its dual variant, $\mathsf{dual\text{-}snPRF\text{-}ODH}$. Both have been established by Brendel et al. [17] to hold for HMAC in the random oracle model under the strong Diffie–Hellman assumption.

Definition 2.3. ($\mathsf{snPRF\text{-}ODH}$ *and* $\mathsf{dual\text{-}snPRF\text{-}ODH}$ *assumptions*) Let $\lambda \in \mathbb{N}$, $\mathbb{G}$ be a cyclic group of prime order q with generator g and $\mathsf{PRF}\colon \mathbb{G} \times \{0,1\}^* \to \{0,1\}^\lambda$ be a pseudorandom function.

We define the $\mathsf{snPRF\text{-}ODH}$ security game as follows.

1. The challenger samples $b \leftarrow_\$ \{0,1\}$, $u, v \leftarrow_\$ \mathbb{Z}_q$, and provides $\mathbb{G}$, g, g^u, and g^v to $\mathcal{A}$, who responds with a challenge label $x^\star$.
2. The challenger computes $y_0^\star = \mathsf{PRF}(g^{uv}, x^\star)$ and samples $y_1^\star \leftarrow_\$ \{0,1\}^\lambda$ uniformly at random, providing $y_b^\star$ to $\mathcal{A}$.
3. $\mathcal{A}$ may query a pair (S, x), on which the challenger first ensures that $S \notin \mathbb{G}$ or $(S, x) = (g^v, x^\star)$ and, if so, returns $y \leftarrow \mathsf{PRF}(S^u, x)$.
4. Eventually, $\mathcal{A}$ stops and outputs a guess $b' \in \{0,1\}$.

We define the snPRF-ODH advantage function as

$$\mathsf{Adv}^{\mathsf{snPRF\text{-}ODH}}_{\mathsf{PRF},\mathbb{G},\mathcal{A}} := 2 \cdot \Pr[b' = b] - 1.$$

We define the dual variant of the assumption, dual-snPRF-ODH, as the snPRF-ODH assumption for a function $\mathsf{PRF}\colon \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\lambda$ with swapped inputs, keyed with a group element in the second input and taking the label as first input.

3. The TLS 1.3 Handshake Protocol

In this section we describe the TLS 1.3 handshake protocol modes, specifically the full one round-trip time (1-RTT) handshake, depicted on the left-hand side of Fig. 1, and the combined zero round-trip time (0-RTT) and pre-shared key handshake, depicted on the right-hand side of in Fig. 1. Our focus in Fig. 1 and throughout the paper is on the cryptographic aspects of the TLS 1.3 handshake. As such, we omit many other components of the protocol, including most hello extensions, aspects of version and algorithm negotiation, post-handshake messages, the record layer protocol, and the alert protocol.

In TLS 1.3, the 1-RTT and PSK handshakes are divided into two distinct phases: a *key exchange phase*, where the client and the server exchange `Hello` messages to indicate support for different cryptographic options and use the selected parameters to generate key exchange material; and an *authentication phase*, where the client and the server exchange `CertificateVerify` and `Finished` messages, authenticating each other using long-term asymmetric (or symmetric) values. Figure 2 illustrates the key schedule of TLS 1.3, Table 1 lists abbreviations for messages and keys used throughout the paper, and Table 2 details some of the computations and inputs.

3.1. *Key-Exchange Phase*

The *key exchange phase* consists of the exchange of `ClientHello` (`CH`) and `ServerHello` (`SH`) messages, during which parameters are negotiated and the core key exchange is performed, using either Diffie–Hellman key exchange or based on a pre-shared symmetric key.

`ClientHello`. The client begins by sending the `ClientHello` message, which contains r_c (a randomly-sampled 256-bit nonce value), as well as version and algorithm negotiation information.

Attached to the `ClientHello` is the `KeyShare` (`CKS`) extension which contains public key shares for the key exchange. Other extensions are present for further algorithm and parameter negotiation. (Note that our analysis in Sects. 5 and 6 does not consider the negotiation of cryptographic values (such as pre-shared keys or (EC)DHE groups) or handshake modes, but instead our analysis considers each handshake mode and ciphersuite combination in isolation. This can be seen in Fig. 1, e.g., the `CKS` message contains only a single (EC)DHE key share.)

If a pre-shared secret has been established between the client and the server (either in a previous handshake or via some out-of-band mechanism) the client may include

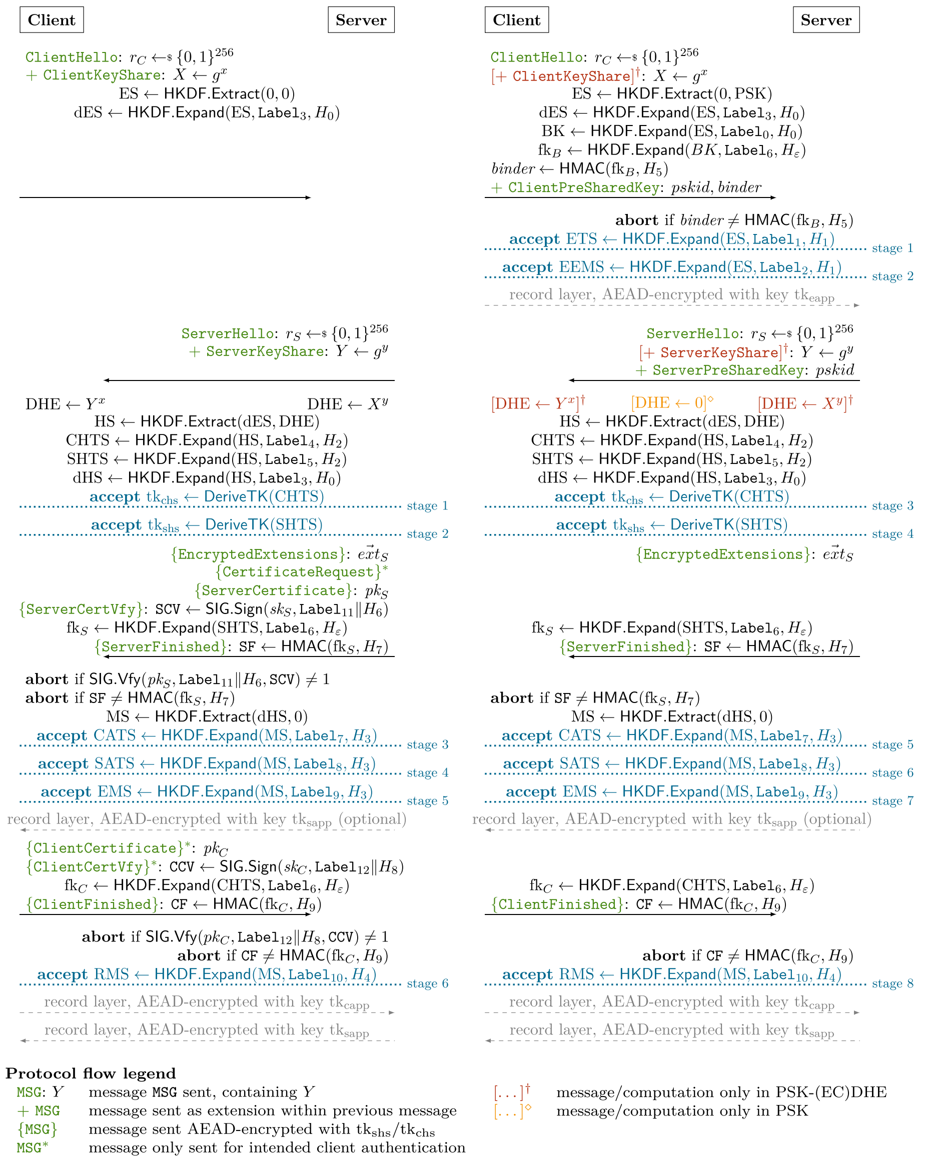
Client
Server
ClientHello: $r_C \leftarrow_{\$} \{0,1\}^{256}$
+ ClientKeyShare: $X \leftarrow g^x$
ES ← HKDF.Extract(0, 0)
dES ← HKDF.Expand(ES, Label$_3$, H_0)
ServerHello: $r_S \leftarrow_{\$} \{0,1\}^{256}$
+ ServerKeyShare: $Y \leftarrow g^y$
DHE ← Y^x
DHE ← X^y
HS ← HKDF.Extract(dES, DHE)
CHTS ← HKDF.Expand(HS, Label$_4$, H_2)
SHTS ← HKDF.Expand(HS, Label$_5$, H_2)
dHS ← HKDF.Expand(HS, Label$_3$, H_0)
accept tk$_{chs}$ ← DeriveTK(CHTS)
stage 1
accept tk$_{shs}$ ← DeriveTK(SHTS)
stage 2
{EncryptedExtensions}: $\vec{ext}_S$
{CertificateRequest}*
{ServerCertificate}: pk_S
{ServerCertVfy}: SCV ← SIG.Sign(sk_S, Label$_{11}$‖H_6)
fk$_S$ ← HKDF.Expand(SHTS, Label$_6$, H_ε)
{ServerFinished}: SF ← HMAC(fk$_S$, H_7)
abort if SIG.Vfy(pk_S, Label$_{11}$‖H_6, SCV) ≠ 1
abort if SF ≠ HMAC(fk$_S$, H_7)
MS ← HKDF.Extract(dHS, 0)
accept CATS ← HKDF.Expand(MS, Label$_7$, H_3)
stage 3
accept SATS ← HKDF.Expand(MS, Label$_8$, H_3)
stage 4
accept EMS ← HKDF.Expand(MS, Label$_9$, H_3)
stage 5
record layer, AEAD-encrypted with key tk$_{sapp}$ (optional)
{ClientCertificate}*: pk_C
{ClientCertVfy}*: CCV ← SIG.Sign(sk_C, Label$_{12}$‖H_8)
fk$_C$ ← HKDF.Expand(CHTS, Label$_6$, H_ε)
{ClientFinished}: CF ← HMAC(fk$_C$, H_9)
abort if SIG.Vfy(pk_C, Label$_{12}$‖H_8, CCV) ≠ 1
abort if CF ≠ HMAC(fk$_C$, H_9)
accept RMS ← HKDF.Expand(MS, Label$_{10}$, H_4)
stage 6
record layer, AEAD-encrypted with key tk$_{capp}$
record layer, AEAD-encrypted with key tk$_{sapp}$
Client
Server
ClientHello: $r_C \leftarrow_{\$} \{0,1\}^{256}$
[+ ClientKeyShare]†: $X \leftarrow g^x$
ES ← HKDF.Extract(0, PSK)
dES ← HKDF.Expand(ES, Label$_3$, H_0)
BK ← HKDF.Expand(ES, Label$_0$, H_0)
fk$_B$ ← HKDF.Expand(BK, Label$_6$, H_ε)
$binder$ ← HMAC(fk$_B$, H_5)
+ ClientPreSharedKey: $pskid$, $binder$
abort if $binder$ ≠ HMAC(fk$_B$, H_5)
accept ETS ← HKDF.Expand(ES, Label$_1$, H_1)
stage 1
accept EEMS ← HKDF.Expand(ES, Label$_2$, H_1)
stage 2
record layer, AEAD-encrypted with key tk$_{eapp}$
ServerHello: $r_S \leftarrow_{\$} \{0,1\}^{256}$
[+ ServerKeyShare]†: $Y \leftarrow g^y$
+ ServerPreSharedKey: $pskid$
[DHE ← Y^x]†
[DHE ← 0]$^\diamond$
[DHE ← X^y]†
HS ← HKDF.Extract(dES, DHE)
CHTS ← HKDF.Expand(HS, Label$_4$, H_2)
SHTS ← HKDF.Expand(HS, Label$_5$, H_2)
dHS ← HKDF.Expand(HS, Label$_3$, H_0)
accept tk$_{chs}$ ← DeriveTK(CHTS)
stage 3
accept tk$_{shs}$ ← DeriveTK(SHTS)
stage 4
{EncryptedExtensions}: $\vec{ext}_S$
fk$_S$ ← HKDF.Expand(SHTS, Label$_6$, H_ε)
{ServerFinished}: SF ← HMAC(fk$_S$, H_7)
abort if SF ≠ HMAC(fk$_S$, H_7)
MS ← HKDF.Extract(dHS, 0)
accept CATS ← HKDF.Expand(MS, Label$_7$, H_3)
stage 5
accept SATS ← HKDF.Expand(MS, Label$_8$, H_3)
stage 6
accept EMS ← HKDF.Expand(MS, Label$_9$, H_3)
stage 7
record layer, AEAD-encrypted with key tk$_{sapp}$ (optional)
fk$_C$ ← HKDF.Expand(CHTS, Label$_6$, H_ε)
{ClientFinished}: CF ← HMAC(fk$_C$, H_9)
abort if CF ≠ HMAC(fk$_C$, H_9)
accept RMS ← HKDF.Expand(MS, Label$_{10}$, H_4)
stage 8
record layer, AEAD-encrypted with key tk$_{capp}$
record layer, AEAD-encrypted with key tk$_{sapp}$
Protocol flow legend
MSG: Y message MSG sent, containing Y
+ MSG message sent as extension within previous message
{MSG} message sent AEAD-encrypted with tk$_{shs}$/tk$_{chs}$
MSG* message only sent for intended client authentication
[...]† message/computation only in PSK-(EC)DHE
[...]$^\diamond$ message/computation only in PSK

Fig. 1. The TLS 1.3 full 1-RTT handshake protocol (left) and the PSK/PSK-(EC)DHE handshake protocol with optional 0-RTT (right). Shorthands are explained in Table 1; the values of context and label inputs (H_*, resp. Label$_*$) and details on the calculation of traffic keys (tk$_*$) can be found in Table 2.

the `PreSharedKey` (`CPSK`) extension, which indicates handshake modes (such as PSK or PSK-(EC)DHE) that the client supports, and a list of pre-shared symmetric identities that map to these PSKs. (As for the `KeyShare` extension, we do not consider negotiation here and only capture the single PSK entry that client and server agree upon.) If `CPSK` is included, the client computes a binder key value BK for each pre-shared key PSK in the list, from that a key fk_B, and a value *binder* $\leftarrow \mathsf{HMAC}(\mathrm{fk}_B, H(\mathtt{CH}^{\dagger}))$ that binds the current `CH` message (truncated to exclude the *binder* value itself) to each PSK, also included in the `CPSK` message and checked by the server. This is captured on the right-hand side of Fig. 1.

Finally, if the client wishes to use the pre-shared secret to send zero-round-trip time (0-RTT) data, the client can indicate this by sending a `EarlyDataIndication` extension. This will indicate to the server that the client will use the first pre-shared secret indicated in the `CPSK` list to derive an early traffic secret (ETS), and early exporter master secret (EEMS), and begin sending encrypted data to the server without first requiring the client to receive `ServerHello` response.

`ServerHello`. The next message in the key-exchange phase is the `ServerHello` (`SH`) message. As in `CH`, the server will randomly sample a 256-bit nonce value r_s. The server picks among the various algorithms and parameters offered by the client and responds with its selections. If `CPSK` was sent, the server decides whether to accept a PSK-based handshake. If so, then the pre-shared key identifier *pskid* associated with the selected PSK is sent in the `PreSharedKey` (`SPSK`) extension. If the server has chosen PSK-(EC)DHE mode (or has rejected the use of PSKs), the server will generate its own (EC)DHE key share $Y \leftarrow g^y$, sending Y in the `KeyShare` (`SKS`) extension attached to `SH`.

At this point, the server has enough information to compute the client handshake traffic secret (CHTS) and server handshake traffic secret (SHTS) values, and uses these to derive client and server handshake traffic keys ($\mathrm{tk}_{\mathrm{chs}}$ and $\mathrm{tk}_{\mathrm{shs}}$, respectively). The first part of Fig. 2 shows the key schedule for deriving these keys. Note that we consider $\mathrm{tk}_{\mathrm{chs}}$ and $\mathrm{tk}_{\mathrm{shs}}$ being derived at the same point in time (namely when the handshake secret HS becomes available), although $\mathrm{tk}_{\mathrm{chs}}$ is in principle only needed a bit later.

The server now begins to encrypt all handshake messages under $\mathrm{tk}_{\mathrm{shs}}$, and any extensions that are not required to establish the server handshake traffic key are sent (and encrypted) in the `EncryptedExtensions` (`EE`) messages.

3.2. *Authentication Phase*

The *authentication phase* now begins. All handshake messages in this phase are encrypted under $\mathrm{tk}_{\mathrm{shs}}$ or $\mathrm{tk}_{\mathrm{chs}}$. In the full 1-RTT handshake, authentication is based on public key certificates; see the left-hand side of Fig. 1. In pre-shared key handshakes (both PSK and PSK-(EC)DHE), the server and client will authenticate each other by relying on a message authentication code applied to the transcript; see the right-hand side of Fig. 1.

Authentication in Full 1-RTT Handshake The server can request public-key-based client authentication by sending a `CertificateRequest` (`CR`) message. The server will authenticate to the client by using the server's long-term public keys. Here, the server

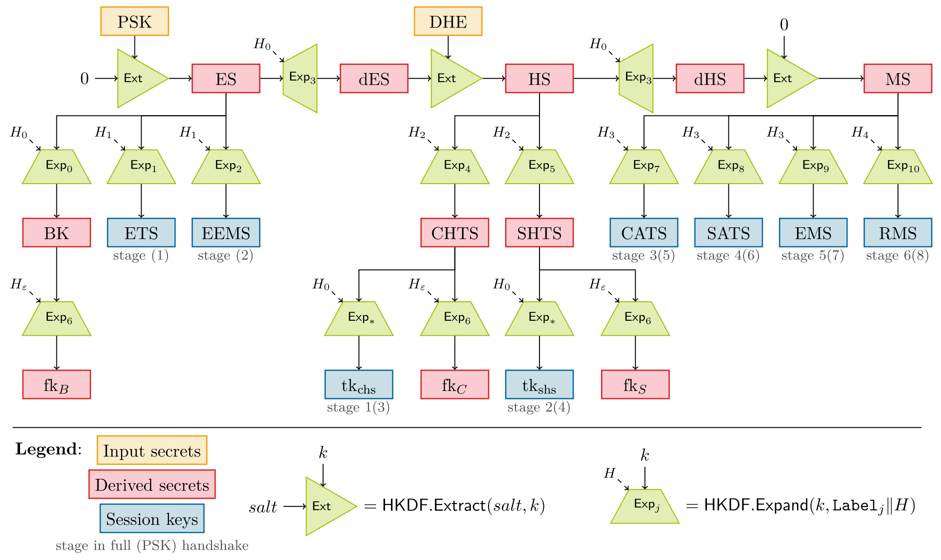

Fig. 2. The TLS 1.3 key schedule. The values of context and label inputs (H_*, resp. $\mathtt{Label}_*$) and details on the calculation of traffic keys (Exp_*) can be found in Table 2.

begins by sending its certificate (carrying its public key) in the `ServerCertificate` (`SCRT`) message. The server then computes `ServerCertificateVerify` authentication value by signing the session hash (which is a continuously updating hash of all messages up to this point in the protocol), then sends it to the client as the `ServerCertificateVerify` message.

Server Key Confirmation and Key Derivation In all handshake modes, the final message that the server sends to the client is the `ServerFinished` (`SF`) message. The server first derives a server finished key fk_S from SHTS and then computes a MAC tag `SF` over the session hash. This value is also encrypted under $\mathrm{tk}_{\mathrm{shs}}$, sending the output ciphertext to the client. At this point, the server is able to compute the client application traffic secret (CATS), the server application traffic secret (SATS), and the exporter master secret (EMS). Figure 2 shows the key schedule for deriving these keys and all other keys in the authentication phase. Now that the server has computed the server application traffic key $\mathrm{tk}_{\mathrm{sapp}}$, it can begin sending encrypted application data to the client without waiting for the final flight of messages from the client, thus achieving a 0.5-RTT handshake.

Client Verification, Authentication, Key Confirmation, and Key Derivation The client, upon receiving these messages, checks that the signature `SCV` (if in full 1-RTT mode) and the MAC `SF` verify correctly. If the server has requested client authentication, the client will begin by sending its digital certificate (carrying its public-key) in the `ClientCertificate` (`CCRT`) message, after which the client will compute its own certificate verify value `CCV` by signing the session hash, then send it to the server as the `CCV` message. The client finally derives the client finished key fk_C from CHTS and uses fk_C to compute a MAC tag `CF` over the session hash.

Server Verification The server will verify the final MAC (`SF`) and optional signature (`SCV`) messages of the client.

Handshake Completion At this point both parties can compute the resumption master secret (RMS) value that can be used as a pre-shared key for session resumption in the future. Both parties can now derive the client application traffic key ($\mathrm{tk}_{\mathrm{capp}}$) and use the record layer for encrypted communication of application data with the resulting keys.

3.3. *NewSessionTicket*

The `NewSessionTicket` message is a post-handshake message in TLS 1.3 which refers to values from the handshake protocol. The `NewSessionTicket` message can be sent by a server to the client (encrypted under a server application traffic key $\mathrm{tk}_{\mathrm{sapp}}$) to allow the client to compute values associated with resumption handshakes, including the PSK used in resumption as well as an identifier to indicate to the server which pre-shared key is being used. The `NewSessionTicket` message contains two fields that are interesting for this purpose:

Table 1. Shorthands for TLS 1.3 messages (in protocol order) and derived keys/values (alphabetical).

Message		Derived key or value	
`CH`	`ClientHello`	BK	Binder Key
`CKS`	`ClientKeyShare`	CHTS/SHTS	Client/Server Handshake Traffic Secret
`CPSK`	`ClientPreSharedKey`	CATS/SATS	Client/Server Application Traffic Secret
`SH`	`ServerHello`	dES/dHS	Derived Early/Handshake Secret
`SKS`	`ServerKeyShare`	ES/HS/MS	Early/Handshake/Master Secret
`SPSK`	`ServerPreSharedKey`	ETS	Early Traffic Secret
`EE`	`EncryptedExtensions`	EEMS/EMS	(Early) Exporter Master Secret
`CR`	`CertificateRequest`	$\mathrm{fk}_B/\mathrm{fk}_C/\mathrm{fk}_S$	Binder/Client/Server Finished Key
`SCRT`	`ServerCertificate`	RMS	Resumption Master Secret
`SCV`	`ServerCertificateVerify`	$\mathrm{tk}_{\mathrm{eapp}}$	Early Application Traffic Key
`SF`	`ServerFinished`	$\mathrm{tk}_{\mathrm{chs}}/\mathrm{tk}_{\mathrm{shs}}$	Client/Server Handshake Traffic Key
`CCRT`	`ClientCertificate`	$\mathrm{tk}_{\mathrm{capp}}/\mathrm{tk}_{\mathrm{sapp}}$	Client/Server Application Traffic Key
`CCV`	`ClientCertificateVerify`		
`CF`	`ClientFinished`		

- `ticket_nonce`, which is used by the client as the salt value to derive the pre-shared key to be used in future handshake for resumption: PSK $\leftarrow$ $\mathsf{HKDF.Expand}$(RMS, "`resumption`", `ticket_nonce`).
- `ticket`, which is an opaque label used to publicly refer to the associated pre-shared key in future `PreSharedKey` messages. In our notation used in Fig. 1, the pre-shared key identifier $pskid$ = `ticket`.

In our analysis, we do not capture this `NewSessionTicket` message, nor the derivation of PSK from RMS, and instead assume that the mapping between PSK and $pskid$ is established in some out-of-band way. In particular, we do not capture transmission of `NewSessionTicket` under a server application traffic key $\mathrm{tk}_{\mathrm{sapp}}$, as it would impact how we consider the usage of SATS. In our analysis, we currently consider SATS an "`external key`", used in an arbitrary symmetric-key protocol. To capture the transmission of `NewSessionTicket`, we would need to capture the use of SATS in deriving $\mathrm{tk}_{\mathrm{sapp}}$ and then establishing PSK. We choose to simplify the analysis by omitting this mechanism and leave this as future work.

4. Multi-Stage Key Exchange Security Model

In order to capture the security of all variants of the TLS 1.3 handshake within a single comprehensive security model, we adopt the multi-stage key exchange model in the version by Günther [61] which combines the original model by Fischlin and Günther [52] with follow-up extensions [38,39,53]. We refer to Günther [61] for an extensive discussion of the model, but recap its core concepts and definitions as well as adaptations for our analysis in the following.

The model follows the classical paradigm for key exchange models of Bellare and Rogaway [25] in the formalism of Brzuska et al. [22,26]. This paradigm captures a strong adversary that controls the network and is able to both passively eavesdrop and

Table 2. Secret, context, and label inputs to the HKDF.Expand resp. authentication functions as well as traffic key calculation in the TLS 1.3 handshake (Fig. 1) and key schedule (Fig. 2).

Secret	Context input	Label input
BK	$H_0 = H(\text{``''})$	$\texttt{Label}_0 = \text{``ext binder''/``res binder''}$
fk_B	$H_\varepsilon = \text{``''}$	$\texttt{Label}_6 = \text{``finished''}$
ETS	$H_1 = H(\texttt{ClientHello})$	$\texttt{Label}_1 = \text{``c e traffic''}$
EEMS	$H_1 = H(\texttt{ClientHello})$	$\texttt{Label}_2 = \text{``e exp master''}$
dES	$H_0 = H(\text{``''})$	$\texttt{Label}_3 = \text{``derived''}$
CHTS	$H_2 = H(\texttt{ClientHello}\|\texttt{ServerHello})$	$\texttt{Label}_4 = \text{``c hs traffic''}$
SHTS	$H_2 = H(\texttt{ClientHello}\|\texttt{ServerHello})$	$\texttt{Label}_5 = \text{``s hs traffic''}$
fk_S	$H_\varepsilon = \text{``''}$	$\texttt{Label}_6 = \text{``finished''}$
dHS	$H_0 = H(\text{``''})$	$\texttt{Label}_3 = \text{``derived''}$
CATS	$H_3 = H(\texttt{ClientHello}\|\ldots\|\texttt{ServerFinished})$	$\texttt{Label}_7 = \text{``c ap traffic''}$
SATS	$H_3 = H(\texttt{ClientHello}\|\ldots\|\texttt{ServerFinished})$	$\texttt{Label}_8 = \text{``s ap traffic''}$
EMS	$H_3 = H(\texttt{ClientHello}\|\ldots\|\texttt{ServerFinished})$	$\texttt{Label}_9 = \text{``exp master''}$
fk_C	$H_\varepsilon = \text{``''}$	$\texttt{Label}_6 = \text{``finished''}$
RMS	$H_4 = H(\texttt{ClientHello}\|\ldots\|\texttt{ClientFinished})$	$\texttt{Label}_{10} = \text{``res master''}$
Auth. value	**Context input**	**Context string (for signatures only)**
binder	$H_5 = H(\texttt{ClientHello}^\dagger)$	
SCV	$H_6 = H(\texttt{ClientHello}\|\ldots\|\texttt{ServerCert})$	$\texttt{Label}_{11} = \text{``TLS 1.3, server CertificateVerify''}$
SF	$H_7 = H(\texttt{ClientHello}\|\ldots\|\texttt{ServerCertVfy})$	
CCV	$H_8 = H(\texttt{ClientHello}\|\ldots\|\texttt{ClientCert})$	$\texttt{Label}_{12} = \text{``TLS 1.3, client CertificateVerify''}$
CF	$H_9 = H(\texttt{ClientHello}\|\ldots\|\texttt{ClientCertVfy}^*)$	
Traffic key calculation		
$\mathrm{tk}_{\mathrm{eapp}}/\mathrm{tk}_{\mathrm{chs}}/\mathrm{tk}_{\mathrm{shs}}/\mathrm{tk}_{\mathrm{capp}}/\mathrm{tk}_{\mathrm{sapp}} = (key, iv) =$ DeriveTK(ETS/CHTS/SHTS/CATS/SATS) where DeriveTK(Secret) $=$ (HKDF.Expand(Secret, $\text{``key''}$, $H(\text{``''})$, L_k), HKDF.Expand(Secret, $\text{``iv''}$, $H(\text{``''})$, L_{iv})) with L_k/L_{iv} indicating the key/iv length of the negotiated AEAD scheme		

The actual label input to HKDF.Expand is the concatenation of the hash length (in bytes), the string "tls13 ", Label, and the given context value. HKDF.Expand is then called on the corresponding secret, this augmented label, and the desired output length. ClientCertVfy* is only included in case of client authentication. ClientHello† indicates a truncated version of ClientHello which excludes the *binder* value itself. Signatures in SCV and CCV are computed over the concatenation of a constant (0x20 repeated 64 times), the label as context information, a separating 0 byte, and the context value

to actively modify the communication across multiple sessions of the key exchange protocol (spawning them via a NewSession oracle and directing communication via a Send oracle). The adversary is further allowed to expose the long-term secrets of interacting honest parties (via a Corrupt oracle) as well as the session keys in some protocol runs (through a Reveal oracle). Basic security then demands that such adversary cannot distinguish the real established session key in some uncompromised ("fresh") session from a random one (through a Test oracle).

The multi-stage key exchange model now extends the basic key exchange setting by capturing protocols that derive a series of session keys in multiple *stages*. Each stage is associated with particular security properties, steering admissibility of certain adversarial actions for that stage and under which conditions the key of this stage is considered fresh. These security properties model the following aspects:

Authentication. Our model distinguishes between *unauthenticated* stages, *unilaterally authenticated* stages where only the responder (the server in TLS 1.3) authenticates, and *mutually authenticated* stages where both peers authenticate. We treat the authentication of each stage *individually* and consider *concurrent* executions of different authentication modes of the same protocol. The identities of communication partners may be learned only during the execution of the protocol (e.g., through exchanged certificates), which we implement through *post-specified peers* following Canetti and Krawczyk [34]. Our model demands a strong notion of security for sessions with unauthenticated peers, namely that such sessions achieve key secrecy when receiving their messages from an honest session (identified via a *contributive identifier*), independent of whether that honest peer session later becomes partnered. Moreover, the authentication level of some stage may be raised with acceptance of a later stage, e.g., from unauthenticated to unilaterally or even mutually authenticated. This may happen for instance if a party later signs previously transmitted data, as in case of TLS 1.3. We capture this by allowing a protocol to specify the authentication level for each acceptance stage, as well as at which later stage(s) that level increases. Note that we capture authentication *implicitly* through key secrecy (i.e., keys are only known to the intended peer session) but do not prove explicit authentication (i.e., the existence of a partnered session). The SIGMA design [72], on which the main TLS 1.3 handshake is based, ensures explicit authentication. de Saint Guilhem et al. [42] give a generic argument that explicit authentication follows from implicit key secrecy (which is shown for TLS 1.3 in this article) and key confirmation [55].

Forward secrecy. We capture the usual notion of forward secrecy, which ensures that accepted session keys remain secure after a long-term secret compromise. In a multi-stage key exchange protocol, forward secrecy may however be reached only from a certain stage on (e.g., due to mixing-in forward-secret key material). The model hence treats *stage-j forward secrecy*, indicating that keys from stage j on are forward secret.

Key usage. Some stage keys might be used internally in the key exchange protocol, e.g., in the case of TLS 1.3 the handshake key is used to encrypt part of the key exchange communication. We distinguish the usage of keys as *internal* when used within the key exchange, and *external* when exclusively used outside of the key exchange (e.g., to encrypt application data). In the former case, our model ensures that tested real-or-random keys are accordingly used in subsequent key exchange steps, and pauses the protocol execution to enable testing of those keys. We note that the declaration of whether a key is internal or external is a parameter to the model, and becomes a part of the protocol description and its security guarantees.

Public or pre-shared keys. Our multi-stage model comes in two flavors that capture both the regular, public-key case (abbreviated as pMSKE) of long-term keys being public/secret key pairs (as in the TLS 1.3 full handshake) as well as the pre-shared–

secret case (abbreviated sMSKE) case where pre-shared symmetric keys act as long-term secrets (as in the TLS 1.3 resumption handshake).

Replayability. For 0-RTT key establishment, key exchange protocols (including TLS 1.3) regularly give up strong replay protection guarantees, in the sense that client (initiator) messages can be replayed to several server (responder) sessions. We capture this in our model by distinguishing between *replayable* (0-RTT) and regular *non-replayable* stages, taking potential replays into account for the former while still requiring key secrecy. Determining the replay type of a stage is again a parameter to the model and must be specified as part of the protocol description resp. the security claim.

We note that former variants of multi-stage key exchange models including [61] further differentiated whether the compromise of some stage's key affects the security of other stages' keys under the notion of *key (in)dependence*. Here, we always demand such compromise never affects other stages' keys as the desirable goal, i.e., we postulate key $\underline{\text{in}}$dependence and reduce the model's complexity by incorporating this property straight into the model. As we will see, TLS 1.3 always achieves this property due to clean key separation in the key scheduling and already did so in earlier draft versions [38,39,53].

Secret Compromise Paradigm We follow the paradigm of the Bellare–Rogaway model [25], focusing on the leakage of long-term secret inputs and session key outputs of the key exchange, but not on internal values within the execution of a session. This contrasts to some extent with the model by Canetti and Krawczyk [33] resp. LaMacchia et al. [79] which include a "session state reveal" resp. "ephemeral secret reveal" query that allows accessing internal variables of the session execution.

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

- *Long-term keys* (such as the signing keys of the server or client, but also their pre-shared keys), since long-term values have the potential to be compromised, and this is necessary to model forward secrecy; it is captured in our model by the $\mathsf{Corrupt}$ query.
- *Session keys* (such as the various traffic encryption keys or the derived resumption or exporter secrets), since these are outputs of the key exchange and are used beyond this protocol for encryption, later resumption, or exporting of keying material; this is modeled by the Reveal query.

We do not permit the leakage of:

- *Ephemeral secrets / randomness* (such as the randomness in a signature algorithm or ephemeral Diffie–Hellman exponents); this is disallowed since TLS 1.3 is not designed to be secure if these values are compromised.
- *Internal values / session state* (e.g., internally computed master secrets or MAC keys); this is disallowed since TLS 1.3 is not designed to be secure if these values are compromised.

Comparison with Previous Multi-stage Key Exchange Models Compared to the original MSKE model of Fischlin and Günther [52], the most notable changes in our model are the addition which models upgradeable authentication and accommodating both public and pre-shared symmetric keys for authentication. We also do not track whether keys are independent or not, as all keys established in TLS 1.3 satisfy key independence (unlike in the analysis of QUIC in [52]). Key usage (internal versus external) and replayability were introduced to MSKE by [53].

4.1. *Syntax*

In our model, we explicitly separate some *protocol-specific* properties (as, e.g., various authentication flavors) from *session-specific* properties (as, e.g., the state of a running session). We represent protocol-specific properties as a vector (M, AUTH, FS, USE, REPLAY) that captures the following:

- $\mathsf{M} \in \mathbb{N}$: the number of stages (i.e., the number of keys derived). (We fix a maximum stage M only for ease of notation. Note that M can be arbitrarily large in order to cover protocols where the number of stages is not bounded a-priori. Also note that stages and session key derivations may be "back to back," without further protocol interactions between parties.)
- $\mathsf{AUTH} \subseteq \{((u_1, m_1), \ldots, (u_\mathsf{M}, m_\mathsf{M})) \mid u_j, m_j \in \{1, \ldots, \mathsf{M}, \infty\}\}$: a set of vectors of pairs, each vector encoding a supported scheme for authentication and authentication upgrades, for each stage. For example, the i-th entry (u_i, m_i) in a vector says that the session key in stage i initially has the default *unauthenticated* level, i.e., provides no authentication for either communication partner, then at stage u_i becomes *unilaterally authenticated* and thus authenticates only the responder (server), and becomes *mutually authenticated* to authenticate both communication partners at stage m_j. Note that we allow for example $u_i = i$ (or even $u_i = m_i = i$) such that the session key is immediately unilaterally (resp. mutually) authenticated when derived. Entries in each pair must be non-decreasing, and $u_i = \infty$ or $m_i = \infty$ denotes that unilateral, resp. mutual, authentication is never reached for stage i.
- $\mathsf{FS} \in \{1, \ldots, \mathsf{M}, \infty\}$: the stage $j = \mathsf{FS}$ from which on keys are forward secret (or ∞ in case of no forward secrecy). (A more general multi-stage key exchange model could have a vector tracking specifically which subset of stage keys have forward secrecy. We do not need such generality since forward secrecy is monotonic in TLS 1.3.)
- $\mathsf{USE} \in \{\mathsf{internal}, \mathsf{external}\}^\mathsf{M}$: the usage indicator for each stage, where USE_i indicates the usage of the stage-i key. Here, an internal key is used within the key exchange protocol (but possibly also externally), whereas an external key must not be used within the protocol, making the latter potentially amenable to generic composition (cf. Sect. 7.3). As shorthand notation, we, e.g., write $\mathsf{USE} = (\mathsf{internal} : \{1, 4\}, \mathsf{external} : \{2, 3, 5\})$ to indicate that usage of keys in stage 1 and 4 is internal, and external for the other stages.
- $\mathsf{REPLAY} \in \{\mathsf{replayable}, \mathsf{nonreplayable}\}^\mathsf{M}$: the replayability indicator for each stage, where REPLAY_i indicates whether the i-th stage is replayable in the sense that an adversary can easily force identical communication and thus identical session identifiers and keys in this stage (e.g., by re-sending the same data in 0-RTT stages). Note that the adversary, however, should still not be able to distinguish such a

replayed key from a random one. We remark that, from a security viewpoint, the usage of replayable stages should ideally be limited, although such stages usually come with an efficiency benefit. We use the same shorthand notation as for USE; e.g., $\mathsf{REPLAY} = (\mathsf{nonreplayable} : \{1, 2, 3\})$ indicates that all three stages are non-replayable.

We denote by $\mathcal{U}$ the set of *identities* (or *users*) used to model the participants in the system, each identified by some $U \in \mathcal{U}$. *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 $\mathsf{label} = (U, V, n)$ indicates the n-th local session of identity U (the session *owner*) with V as the intended communication *partner*.

In the public-key variant of the model (pMSKE), each identity U is associated with a certified *long-term* public key pk_U and secret key sk_U. In the pre-shared secret setting (sMSKE), a session instead holds an identifier $\mathsf{pssid} \in \{0, 1\}^*$ for the pre-shared secret $\mathsf{pss} \in \mathcal{P}$ (from some pre-shared secret space $\mathcal{P}$) used. The challenger maintains maps $\mathsf{pss}_{U,V} : \{0, 1\}^* \to \mathcal{P}$ mapping an identifier to the corresponding secret shared by parties U and V, where U uses that secret (only) in the initiator role and V (only) in the responder role, and for any user U, a pre-shared secret identifier pssid uniquely identifies the peer identity V it is shared with. (Requiring a fixed role in which a pre-shared key can be used by either peer avoids the Selfie attack [1,44].)

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

- $\mathsf{label} \in \mathsf{LABELS}$: the unique (administrative) session label
- $\mathsf{id} \in \mathcal{U}$: the identity of the session owner
- $\mathsf{pid} \in \mathcal{U} \cup \{*\}$: the identity of the intended communication partner, where the distinct wildcard symbol '$*$' stands for "currently unknown identity" but can be later 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
- $\mathsf{auth} \in \mathsf{AUTH}$: the intended authentication type vector from the set of supported authentication properties AUTH, where auth_i indicates the authentication level pair for stage i, and $\mathsf{auth}_{i,j}$ its j-th entry
- $\mathsf{pssid} \in \{0, 1\}^* \cup \{\bot\}$: In the pre-shared secret (sMSKE) variant the identifier for the pre-shared secret (i.e., $\mathsf{pss}_{\mathsf{id},\mathsf{pid}}$ if $\mathsf{role} = \mathsf{initiator}$, else $\mathsf{pss}_{\mathsf{pid},\mathsf{id}}$) to be used in the session; can be initialized with $\bot$ if $\mathsf{pid} = *$ is unknown and then must be set (once) when pid is set
- $\mathsf{st}_\mathsf{exec} \in (\mathsf{RUNNING} \cup \mathsf{ACCEPTED} \cup \mathsf{REJECTED})$: the state of execution [$\mathsf{running}_0$], where $\mathsf{RUNNING} = \{\mathsf{running}_i \mid i \in \mathbb{N} \cup \{0\}\}$, $\mathsf{ACCEPTED} = \{\mathsf{accepted}_i \mid i \in \mathbb{N}\}$, $\mathsf{REJECTED} = \{\mathsf{rejected}_i \mid i \in \mathbb{N}\}$; set to $\mathsf{accepted}_i$ in the moment a session accepts the i-th key, to $\mathsf{rejected}_i$ when the session rejects that key (a session may continue after rejecting in a stage (this models, e.g., servers rejecting 0-RTT data from a client, but continuing with the remaining handshake)), and to $\mathsf{running}_i$ when a session continues after accepting the i-th key
- $\mathsf{stage} \in \{0, \ldots, \mathsf{M}\}$: 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, set once (and only) upon acceptance in that stage
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\bot\})^{\mathsf{M}}$: cid_i $[\bot]$ indicates the contributive identifier in stage i, may be set several times until acceptance in that stage
- $\mathsf{key} \in (\{0,1\}^* \cup \{\bot\})^{\mathsf{M}}$: key_i $[\bot]$ indicates the established session key in stage i, set once upon acceptance in that stage
- $\mathsf{st}_{\mathsf{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}^{\mathsf{M}}$: $\mathsf{st}_{\mathsf{key},i}$ $[\mathsf{fresh}]$ indicates the state of the session key in stage i
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^{\mathsf{M}}$: test indicator tested_i $[\mathsf{false}]$, where true means that key_i has been tested
- $\mathsf{corrupted} \in \{0, \dots, \mathsf{M}, \infty\}$: corruption indicator $[\infty]$ holding the stage the session was in when a $\mathsf{Corrupt}$ was issued to its owner or intended partner, including the value 0 if the corruption had taken place before the session started, and ∞ if none of the parties is corrupted

By convention, adding a not fully specified tuple $(\mathsf{label}, \mathsf{id}, \mathsf{pid}, \mathsf{role}, \mathsf{auth})$ resp. $(\mathsf{label}, \mathsf{id}, \mathsf{pid}, \mathsf{role}, \mathsf{auth}, \mathsf{pssid})$ to $\mathsf{List}_{\mathsf{S}}$ sets all other entries to their default value. As shorthands, for some tuple with (unique) label label in $\mathsf{List}_{\mathsf{S}}$ we furthermore write $\mathsf{label}.X$ for that tuple's element X and $\mathsf{label}.(X, Y, Z)$ for the vector (X, Y, Z) of that tuple's elements X, Y, and Z.

We define two distinct sessions label and label' to be *partnered* in stage i if both sessions hold the same session identifier in that stage, i.e., $\mathsf{label}.\mathsf{sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$, and require for correctness that two sessions having a non-tampered joint execution are partnered in all stages upon acceptance.

Our security model treats corruption of long-term secrets (secret keys for pMSKE, pre-shared secrets for sMSKE). While the effects of such compromises on sessions may differ in each setting, we broadly consider the derived keys of some session to be revealed if, in the public-key setting (pMSKE), the owner or peer secret key is compromised, or in the pre-shared secret setting (sMSKE), if the pre-shared secret used for that session is compromised. Forward secrecy comes into play when determining if keys derived prior to the long-term secret corruption are affected, too. In more precise notation, we say a session label is *corrupted* if

- for pMSKE, the session's owner $\mathsf{label}.\mathsf{id}$ or intended communication partner $\mathsf{label}.\mathsf{pid}$ is corrupted (i.e., $\{\mathsf{label}.\mathsf{id}, \mathsf{label}.\mathsf{pid}\} \cap \mathsf{corrupted} \neq \emptyset$), resp.
- for sMSKE, the used pre-shared secret is corrupted (i.e., $(\mathsf{label}.\mathsf{id}, \mathsf{label}.\mathsf{pid}, \mathsf{label}.\mathsf{pssid}) \in \mathsf{corrupted}$, the set of corrupted users) if $\mathsf{label}.\mathsf{role} = \mathsf{initiator}$, resp. $(\mathsf{label}.\mathsf{pid}, \mathsf{label}.\mathsf{id}, \mathsf{label}.\mathsf{pssid}) \in \mathsf{corrupted}$ if $\mathsf{label}.\mathsf{role} = \mathsf{responder}$.

Upgradable Authentication We capture that the authentication level of some stage may increase, possibly twice, with acceptance of a later stage through a per-stage vector in the authentication level matrix. When capturing security, our model however needs to carefully consider the interaction of authentication and corruptions (somewhat similar to what one might be used to for forward secrecy). More precisely, the authentication guarantee of some stage i *after its acceptance* can only step up (in some later stage $j > i$) if the involved parties are not corrupted by the time stage j accepts. Otherwise, the adversary may have impersonated the party up to the unauthenticated stage i and now post-authenticates as the party after corruption in stage j. This would effectively mean that the adversary has been in full control of the session and may thus know the session key of stage i.

We capture the upgrade by defining the *rectified authentication level* $\mathsf{rect\_auth}_i$ of some stage i in a session with intended authentication vector auth, consisting of pairs $(\mathsf{auth}_{i,1}, \mathsf{auth}_{i,2})$ describing the stage in which the i-th session key gets unilaterally and mutually authenticated, with corruption indicator $\mathsf{corrupted}$, and with current execution stage stage as follows:

$$\mathsf{rect\_auth}_i := \begin{cases} \mathsf{mutual} & \text{if } \mathsf{stage} \geq \mathsf{auth}_{i,2} \text{ and } \mathsf{corrupted} \geq \mathsf{auth}_{i,2} \\ \mathsf{unilateral} & \text{if } \mathsf{stage} \geq \mathsf{auth}_{i,1} \text{ and } \mathsf{corrupted} \geq \mathsf{auth}_{i,1} \\ \mathsf{unauth} & \text{otherwise} \end{cases}$$

This encodes that authentication level of stage i is upgraded (to $\mathsf{unilateral}$ or mutual) when reaching stage $\mathsf{auth}_{i,1}$, resp. $\mathsf{auth}_{i,2}$, only if no corruption affected this session prior to these stages ($\mathsf{auth}_{i,1}$, resp. $\mathsf{auth}_{i,2}$).

4.2. *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. Our adversary model further reflects the advanced security aspects in multi-stage key exchange as outlined above. We conveniently capture admissibility of adversarial interactions and conditions where the adversary trivially loses (such as both revealing and testing the session key in partnered sessions) via a flag lost (initialized to false).

The adversary interacts with the protocol via the following queries.

- $\mathsf{NewSecret}(U, V, \mathsf{pssid})$: This query is only available in the pre-shared secret (sMSKE) variant. Generates a fresh secret with identifier pssid shared between parties U and V, to be used by U in the initiator role and by V in the responder role. If $\mathsf{pss}_{U,V}(\mathsf{pssid})$ is already set, return $\perp$ to ensure uniqueness of pssid identifiers between two parties in these roles. Otherwise, sample $\mathsf{pss} \leftarrow_\$ \mathcal{P}$ uniformly at random from the protocol's pre-shared secret space $\mathcal{P}$ and define $\mathsf{pss}_{U,V}(\mathsf{pssid}) := \mathsf{pss}$.

- $\mathsf{NewSession}(U, V, \mathsf{role}, \mathsf{auth}[, \mathsf{pssid}])$: Creates a new session with a (unique) new label label for owner identity $\mathsf{id} = U$ with role role, having $\mathsf{pid} = V$ as intended partner (potentially unspecified, indicated by $V = *$) and aiming at authentication type auth.
  In the pre-shared secret (sMSKE) variant, the additional parameter pssid identifies the pre-shared secret to be used, namely $\mathsf{pss}_{U,V}(\mathsf{pssid})$ if $\mathsf{role} = \mathsf{initiator}$, resp. $\mathsf{pss}_{V,U}(\mathsf{pssid})$ if $\mathsf{role} = \mathsf{responder}$. The identifier might be unspecified at this point (indicated by $\mathsf{pssid} = \perp$) and may then be set later by the protocol once.

Add $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth})$, resp. $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, \mathsf{pssid})$, to List_S. If label is corrupted, set $\mathsf{label.corrupted} \leftarrow 0$. This encodes the information that the session is corrupt right from the beginning. Return label.

- $\mathsf{Send}(\mathsf{label}, m)$: Sends a message m to the session with label label.
  If there is no tuple with label label in List_S, return $\perp$. Otherwise, run the protocol on behalf of U on message m and return the response and the updated state of execution $\mathsf{label.st}_\mathsf{exec}$. As a special case, if $\mathsf{label.role} = \mathsf{initiator}$ and $m = \mathsf{init}$, the protocol is initiated (without any input message).
  If, during the protocol execution, the state of execution changes to $\mathsf{accepted}_i$, 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 an internal key, before it may be used immediately in the response and thus cannot be tested anymore to prevent trivial distinguishing attacks. It furthermore allows the adversary to corrupt long-term keys in a fine-grained manner after any acceptance of a key.
  If the state of execution changes to $\mathsf{label.st}_\mathsf{exec} = \mathsf{accepted}_i$ for some i and there is a partnered session $\mathsf{label}' \neq \mathsf{label}$ in List_S (i.e., $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$) with $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$, then set $\mathsf{label.tested}_i \leftarrow \mathsf{true}$ and (only if $\mathsf{USE}_i = \mathsf{internal}$) $\mathsf{label.key}_i \leftarrow \mathsf{label}'.\mathsf{key}_i$. This ensures that, if the partnered session has been tested before, subsequent Test queries for the session are answered accordingly and, in case it is used internally, this session's key key_i is set consistently. (Note that for internal keys 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.)
  If the state of execution changes to $\mathsf{label.st}_\mathsf{exec} = \mathsf{accepted}_i$ for some i and the session label is corrupted, then set $\mathsf{label.st}_{\mathsf{key},i} \leftarrow \mathsf{revealed}$.

- $\mathsf{Reveal}(\mathsf{label}, i)$: Reveals the session key $\mathsf{label.key}_i$ of stage i in the session with label label.
  If there is no session with label label in List_S or $\mathsf{label.stage} < i$, then return $\perp$. Otherwise, set $\mathsf{label.st}_{\mathsf{key},i}$ to $\mathsf{revealed}$ and provide the adversary with $\mathsf{label.key}_i$.

- $\mathsf{Corrupt}(U)$ or $\mathsf{Corrupt}(U, V, \mathsf{pssid})$: The first query is only used in the public-key (pMSKE) variant, the second query only in the pre-shared secret (sMSKE) variant. Provide the adversary with the corresponding long-term secret, i.e., sk_U (pMSKE), resp. $\mathsf{pss}_{U,V}(\mathsf{pssid})$ (sMSKE). Add to the set of corrupted entities $\mathsf{corrupted}$ the user U (for pMSKE), resp. (for sMSKE) the global pre-shared secret identifier (U, V, pssid).

Record the time of corruption in each session label with $\mathsf{label.id} = U$ or $\mathsf{label.pid} = U$ (pMSKE), resp. with $\mathsf{label.(role, id, pid, pssid)} \in \{(\mathsf{initiator}, U, V, \mathsf{pssid}), (\mathsf{responder}, V, U, \mathsf{pssid})\}$ (sMSKE), by setting $\mathsf{label.corrupted} \leftarrow \mathsf{label.stage}$ (unless $\mathsf{label.corrupted} \neq \infty$ already, in which case corruption had taken place earlier such that we leave the value unchanged).
In the non-forward-secret case, for each such session label and for all $i \in \{1, \ldots, \mathsf{M}\}$, set $\mathsf{label.st}_{\mathsf{key},i}$ to $\mathsf{revealed}$. I.e., all (previous and future) session keys are considered to be disclosed.
In the case of stage-j forward secrecy, $\mathsf{st}_{\mathsf{key},i}$ of each such session label is instead 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.

- $\mathsf{Test}(\mathsf{label}, i)$: 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 session with label label in $\mathsf{List}_{\mathsf{S}}$ or if $\mathsf{label.st}_{\mathsf{exec}} \neq \mathsf{accepted}_i$ or $\mathsf{label.tested}_i = \mathsf{true}$, return $\perp$. If stage i is internal (i.e., $\mathsf{USE}^i = \mathsf{internal}$) and there is a partnered session label' in $\mathsf{List}_{\mathsf{S}}$ (i.e., $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$) with $\mathsf{label}'.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{accepted}_i$, set the 'lost' flag to $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that keys can only be tested once and, in case of internal keys, if they have just been accepted but not used yet, ensuring also that any partnered session that may have already established this key has not used it. If $\mathsf{label.rect\_auth}_i = \mathsf{unauth}$, or if $\mathsf{label.rect\_auth}_i = \mathsf{unilateral}$ and $\mathsf{label.role} = \mathsf{responder}$, but there is no session label' (for $\mathsf{label} \neq \mathsf{label}'$) in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{label.cid}_i = \mathsf{label}'.\mathsf{cid}_i$, then set $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that having an honest contributive partner is a prerequisite for testing unauthenticated stages, resp. the responder sessions in a unilaterally authenticated stage. (Note that $\mathsf{List}_{\mathsf{S}}$ entries are only created for honest sessions, i.e., sessions generated by $\mathsf{NewSession}$ queries.) The check is based on the uncorrupted authentication level $\mathsf{rect\_auth}_i$ in order to take corruptions between authentication upgrades into account.
Otherwise, set $\mathsf{label.tested}_i$ to true. If the test bit b_{test} is 0, sample a key $K \leftarrow_{\$} \mathcal{D}$ at random from the session key distribution $\mathcal{D}$. If $b_{\mathsf{test}} = 1$, let $K \leftarrow \mathsf{label.key}_i$ be the real session key. If $\mathsf{USE}_i = \mathsf{internal}$ (i.e., the tested i-th key is indicated as being used internally), set $\mathsf{label.key}_i \leftarrow K$; in other words, when $b_{\mathsf{test}} = 0$, we replace an *internally* used session key by the random and independent test key K which is also used for consistent future usage *within* the key exchange protocol. In contrast, *externally used* session keys are not replaced by random ones, the adversary only receives the real (in case $b_{\mathsf{test}} = 1$) or random (in case $b_{\mathsf{test}} = 0$) key. This distinction between internal and external keys for Test queries emphasizes that external keys are not supposed to be used within the key exchange (and hence there is no need to register the tested random key in the protocol's session key field), while internal keys will be used (and hence the tested random key must be deployed in the remaining protocol steps for consistency).

Moreover, if there exists a partnered session label' which has also just accepted the i-th key (i.e., $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$ and $\mathsf{label.st_{exec}} = \mathsf{label}'.\mathsf{st_{exec}} = \mathsf{accepted}_i$), then also set $\mathsf{label}'.\mathsf{tested}_i \leftarrow \mathsf{true}$ and (only if $\mathsf{USE}_i = \mathsf{internal}$) $\mathsf{label}'.\mathsf{key}_i \leftarrow \mathsf{label.key}_i$ to ensure consistency (of later tests and (internal) key usage) 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 K.

4.3. *Security of Multi-Stage Key Exchange Protocols*

As in the formalization of the Bellare–Rogaway key exchange model by Brzuska et al. [22,26], we model security according to two games, one for key indistinguishability, and one for session matching. The former is the classical notion of random-looking keys, refined under the term Multi-Stage security according to the advanced security aspects for multi-stage key exchange: (stage-j) forward secrecy, different authentication modes, and replayability. The Match property complements this notion by guaranteeing that the specified session identifiers sid effectively match the partnered sessions, and is likewise adapted to the multi-stage setting.

4.3.1. *Match Security*

The notion of Match security ensures soundness of the session identifiers sid, i.e., that they properly identify partnered sessions in the sense that

1. sessions with the same session identifier for some stage hold the same key at that stage,
2. sessions with the same session identifier for some stage have opposite roles, except for potential multiple responders in replayable stages,
3. sessions with the same session identifier for some stage agree on that stage's authentication level,
4. sessions with the same session identifier for some stage share the same contributive identifier at that stage,
5. sessions are partnered with the intended (authenticated) participant and, for mutual authentication based on pre-shared secrets, share the same key identifier,
6. session identifiers do not match across different stages, and
7. at most two sessions have the same session identifier at any non-replayable stage.

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

Definition 4.1. (Match *security*) Let KE be a multi-stage key exchange protocol with properties (M, AUTH, FS, USE, REPLAY) and $\mathcal{A}$ be a PPT adversary interacting with KE via the queries defined in Sect. 4.2 in the following game $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$:

Setup. In the public-key variant (pMSKE), 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 (pMSKE) and has access to the queries NewSecret, NewSession, Send, Reveal, Corrupt, and Test.

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 holds:

1. There exist two distinct labels $\mathsf{label}, \mathsf{label}'$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ for some stage $i \in \{1, \ldots, \mathsf{M}\}$, but $\mathsf{label.key}_i \neq \mathsf{label}'.\mathsf{key}_i$. (Different session keys in some stage of partnered sessions.)
2. There exist two distinct labels $\mathsf{label}, \mathsf{label}'$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ for some stage $i \in \{1, \ldots, \mathsf{M}\}$, but $\mathsf{label.role} = \mathsf{label}'.\mathsf{role}$ and $\mathsf{REPLAY}_i = \mathsf{nonreplayable}$, or $\mathsf{label.role} = \mathsf{label}'.\mathsf{role} = \mathsf{initiator}$ and $\mathsf{REPLAY}_i = \mathsf{replayable}$. (Non-opposite roles of partnered sessions in non-replayable stage.)
3. There exist two distinct labels $\mathsf{label}, \mathsf{label}'$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ for some stage $i \in \{1, \ldots, \mathsf{M}\}$, but $\mathsf{label.auth}_i \neq \mathsf{label}'.\mathsf{auth}_i$. (Different authentication types in some stage of partnered sessions.) (Observe that Match security ensures agreement on the *intended* authentication levels (including potential upgrades); the *rectified* authentication level in contrast is a technical element of the security model capturing the actual level achieved in light of early corruptions when evaluating Test queries.)
4. There exist two distinct labels $\mathsf{label}, \mathsf{label}'$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ for some stage $i \in \{1, \ldots, \mathsf{M}\}$, 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.)
5. There exist two distinct labels $\mathsf{label}, \mathsf{label}'$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ and $\mathsf{label.sid}_j = \mathsf{label}'.\mathsf{sid}_j \neq \bot$ for stages $i, j \in \{1, \ldots, \mathsf{M}\}$ where $j \leq i$, with $\mathsf{label.role} = \mathsf{initiator}$ and $\mathsf{label}'.\mathsf{role} = \mathsf{responder}$ such that
   - $\mathsf{label.auth}_{j,1} \leq i$ (unilateral authentication), but $\mathsf{label.pid} \neq \mathsf{label}'.\mathsf{id}$, or
   - $\mathsf{label.auth}_{j,2} \leq i$ (mutual authentication), but $\mathsf{label.id} \neq \mathsf{label}'.\mathsf{pid}$ or (only for sMSKE) $\mathsf{label.pssid} \neq \mathsf{label}'.\mathsf{pssid}$.
   (Different intended authenticated partner or (only sMSKE) different key identifiers in mutual authentication.)
6. There exist two (not necessarily distinct) labels $\mathsf{label}, \mathsf{label}'$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_j \neq \bot$ for some stages $i, j \in \{1, \ldots, \mathsf{M}\}$ with $i \neq j$. (Different stages share the same session identifier.)
7. There exist three pairwise distinct labels $\mathsf{label}, \mathsf{label}', \mathsf{label}''$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i = \mathsf{label}''.\mathsf{sid}_i \neq \bot$ for some stage $i \in \{1, \ldots, \mathsf{M}\}$ with $\mathsf{REPLAY}_i = \mathsf{nonreplayable}$. (More than two sessions share the same session identifier in a non-replayable stage.)
   We say KE is Match-*secure* if for all PPT adversaries $\mathcal{A}$ the following advantage function is negligible in the security parameter:

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

4.3.2. *Multi-Stage Security*

The second and core notion, $\mathsf{Multi\text{-}Stage}$ security, captures Bellare–Rogaway-like key secrecy in the multi-stage setting as follows.

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

Setup. The challenger chooses the test bit $b_{\mathsf{test}} \leftarrow_{\$} \{0, 1\}$ at random and sets $\mathsf{lost} \leftarrow \mathsf{false}$. In the public-key variant (pMSKE), it furthermore generates long-term public/private-key pairs for each participant $U \in \mathcal{U}$.

Query. The adversary $\mathcal{A}$ receives the generated public keys (pMSKE) and has access to the queries $\mathsf{NewSecret}$, $\mathsf{NewSession}$, Send, Reveal, $\mathsf{Corrupt}$, and Test. Recall that such queries may set lost to true.

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

Finalize. The challenger sets the 'lost' flag to $\mathsf{lost} \leftarrow \mathsf{true}$ if there exist two (not necessarily distinct) labels label, label' and some stage $i \in \{1, \ldots, \mathsf{M}\}$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$, $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$, and $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$. (Adversary has tested and revealed the key of some stage 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 condition is independent of 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* with properties (M, AUTH, FS, USE, REPLAY) if KE is Match-secure and for all PPT adversaries $\mathcal{A}$ the following advantage function is negligible in the security parameter:

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

5. Security Analysis of the TLS 1.3 Full 1-RTT Handshake

We now come to analyzing the TLS 1.3 full 1-RTT handshake in the public-key multi-stage key exchange (pMSKE) model.

Protocol Properties The full handshake targets the following protocol-specific properties (M, AUTH, FS, USE, REPLAY):

- $\mathsf{M} = 6$: The full 1-RTT handshake consists of six stages deriving, in order: the client and server handshake traffic keys $\mathrm{tk}_{\mathrm{chs}}$ and $\mathrm{tk}_{\mathrm{shs}}$, the client and server application traffic secrets CATS and SATS, the exporter master secret EMS, and the resumption master secret RMS. As shown in Fig. 1, we consider all stages' keys being derived on either side as soon as the relevant main secret (ES, HS, MS) becomes available, despite client/server keys derived in parallel might become active with some delay based on the flow direction.
- $\mathsf{AUTH} = \big\{((3, m), (3, m), (3, m), (4, m), (5, m), (6, m)) \mid m \in \{6, \infty\}\big\}$: The handshake traffic keys $\mathrm{tk}_{\mathrm{chs}}/\mathrm{tk}_{\mathrm{shs}}$ are initially unauthenticated and all keys are uni-

laterally authenticated after stage 3 is reached. With (optional) client authentication, all keys furthermore become mutually authenticated with stage $m = 6$; otherwise they never reach this level, $m = \infty$.

- $\mathsf{FS} = 1$: The full 1-RTT handshake ensures forward secrecy for all keys derived.
- $\mathsf{USE} = (\mathsf{internal} : \{1, 2\}, \mathsf{external} : \{3, 4, 5, 6\})$: The handshake traffic keys are used internally to encrypt the second part of the handshake; all other keys are external.
- $\mathsf{REPLAY} = (\mathsf{nonreplayable} : \{1, 2, 3, 4, 5, 6\})$: The keys of all stages are non-replayable in the full 1-RTT handshake.

Session and Contributive Identifiers As part of the analysis in the pMSKE model, we need to define how session and contributive identifiers are set for each stage during execution of the TLS 1.3 full 1-RTT handshake.

Session identifiers are set upon acceptance of each stage and include a label and all handshake messages up to this point (entering the key derivation):

$$\begin{aligned}
\mathsf{sid}_1 &= (\text{"CHTS"}, && \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}),\\
\mathsf{sid}_2 &= (\text{"SHTS"}, && \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}),\\
\mathsf{sid}_3 &= (\text{"CATS"}, && \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{CR}^*, \mathtt{SCRT}, \mathtt{SCV}, \mathtt{SF}),\\
\mathsf{sid}_4 &= (\text{"SATS"}, && \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{CR}^*, \mathtt{SCRT}, \mathtt{SCV}, \mathtt{SF}),\\
\mathsf{sid}_5 &= (\text{"EMS"}, && \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{CR}^*, \mathtt{SCRT}, \mathtt{SCV}, \mathtt{SF}),\\
\mathsf{sid}_6 &= (\text{"RMS"}, && \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{CR}^*, \mathtt{SCRT}, \mathtt{SCV}, \mathtt{SF}, \mathtt{CCRT}^*, \mathtt{CCV}^*, \mathtt{CF}).
\end{aligned}$$

Here, starred (*) components are present only in mutual authentication mode. Note that we define session identifiers over the *unencrypted* handshake messages.

For the contributive identifiers in stages 1 and 2, client (resp. server) upon sending (resp. receiving) the `ClientHello` and `ClientKeyShare` messages set $\mathsf{cid}_1 = (\text{"CHTS"}, \mathtt{CH}, \mathtt{CKS})$, $\mathsf{cid}_2 = (\text{"SHTS"}, \mathtt{CH}, \mathtt{CKS})$ and later, upon receiving (resp. sending) the `ServerHello` and `ServerKeyShare` messages, extend it to $\mathsf{cid}_1 = (\text{"CHTS"}, \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS})$, $\mathsf{cid}_2 = (\text{"SHTS"}, \mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS})$. All other contributive identifiers are set to $\mathsf{cid}_i = \mathsf{sid}_i$ (for stages $i \in \{3, 4, 5, 6\}$) when the respective session identifier is set.

5.1. *Match Security*

We are now ready to give our formal security results for the TLS 1.3 full 1-RTT handshake, beginning with Match security.

Theorem 5.1. *(Match security of* `TLS1.3 -full- 1RTT`*). The TLS 1.3 full 1-RTT handshake is* Match*-secure with properties* (M, AUTH, FS, USE, REPLAY) *given above. For any efficient adversary $\mathcal{A}$, we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} \le n_s^2 \cdot \frac{1}{q} \cdot 2^{-|nonce|},$$

where n_s is the maximum number of sessions, q is the group order, and $|nonce| = 256$ is the bit length of the nonces.

Recall that Match security is a soundness property of the session identifiers. From our definition of session identifiers above, it follows immediately that partnered sessions agree on the derived key, opposite roles, authentication properties, contributive identifiers, and the respective stages. As in the proof of Match security for `TLS1.3-full-1RTT`, The security bound arises as the birthday bound for two honest sessions choosing the same nonce and group element; this not happening ensures at most two partners share the same session identifier.

We need to show the seven properties of Match security (cf. Definition 4.1).

1. *Sessions with the same session identifier for some stage hold the same key at that stage.*
The session identifiers in each stage include the Diffie–Hellman shares g^x and g^y (through the `CKS` and `SKS` messages, fixing the only key input DHE $= g^{xy}$ to all derived stage keys (recall that PSK $= 0$ in the TLS 1.3 full 1-RTT handshake). Furthermore, for each stage, the session identifier includes all handshake messages that enter the key derivation: for stages 1 and 2 messages up to `SKS`, for stages 3–5 messages up to `SF`, and for stage 6 all messages (up to `CF`). In each stage, the session identifier hence determines *all* inputs to the key derivation, and agreement on it thus ensures agreement on the stage key.
2. *Sessions with the same session identifier for some stage have opposite roles, except for potential multiple responders in replayable stages.*
Assuming at most two sessions share the same session identifier (which we show below), two initiator (client) or responder (server) sessions never hold the same session identifier as they never accept wrong-role incoming messages, and the initial `Hello` messages are typed with the sender's role. There are no replayable stages in the TLS 1.3 full 1-RTT handshake.
3. *Sessions with the same session identifier for some stage agree on that stage's authentication level.*
By definition, the authentication for stages 1–2 and 3–5 are fixed to unauth and unilateral (from stage 3 on), respectively, hence agreed upon by all sessions. For the last stage, the presence of `CR`, `CCRT`, and `CCV` in sid_6 unambiguously determines if, from stage 6 on, keys are mutually authenticated (and unilaterally otherwise).
4. *Sessions with the same session identifier for some stage share the same contributive identifier.*
This holds due to, for each stage i, the contributive identifier cid_i being final and equal to sid_i once the session identifier is set.
5. *Sessions are partnered with the intended (authenticated) participant.*
This case only applies to unilaterally or mutually authenticated stages, which is achieved, possibly retroactively, when reaching stages 3, resp. stage 6 (only if the client authenticates). In the TLS 1.3 full 1-RTT handshake, peer identities are learned through the `Certificate` messages. As we are only concerned with honest client and server sessions for Match security, which will only send certificates attesting their own identity, agreement on `SCRT` ensures agreeing on the

server (responder) identity, and vice versa for `CCRT` and the client (initiator) identity. Such agreement is ensured through including `SCRT` in the session identifier for stage 3 for unilateral authentication, and `SCRT` and `CCRT` for mutual authentication in sid_6: once two sessions reach these stages and agree on sid_3, resp. sid_6, they (retroactively) also agree on the intended (responder, resp. initiator) peer.

6. *Session identifiers are distinct for different stages.*
This holds trivially as each stage's session identifier has a unique label.

7. *At most two sessions have the same session identifier at any non-replayable stage.*
Recall that all session identifiers held by some session include that session's random nonce and Diffie–Hellman share. Therefore, for a threefold collision among session identifiers of honest parties, some session would need to pick the same group element and nonce as one other session (which then may be partnered through a regular protocol run to some third session). The probability for such collision to happen can be bounded from above by the birthday bound $n_s^2 \cdot 1/q \cdot 2^{-|\text{nonce}|}$, where n_s is the maximum number of sessions, q is the group order, and $|\text{nonce}| = 256$ the nonces' bit length. □

5.2. *Multi-Stage Security*

We now come to the core multi-stage security result for the TLS 1.3 full 1-RTT handshake.

Theorem 5.2. *(*$\mathsf{Multi\text{-}Stage}$ *security of* `TLS1.3-full-1RTT`*). The TLS 1.3 full 1-RTT handshake is* $\mathsf{Multi\text{-}Stage}$*-secure with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{FS}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. Formally, for any efficient adversary* $\mathcal{A}$ *against the* $\mathsf{Multi\text{-}Stage}$ *security there exist efficient algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_7$ *such that*

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} \\
&\le 6n_s \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG},\mathcal{B}_2} \\ + n_s \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_4} \\ + 2 \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_6} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_7} \end{array} \right) \end{array} \right)
\end{aligned}$$

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

For the TLS 1.3 full 1-RTT handshake, $\mathsf{Multi\text{-}Stage}$ security essentially follows from two lines of reasoning. First, the (unforgeable) signatures covering (a collision-resistant hash of) the full `Hello` messages ensure that session stages with an authenticated peer share exchanged Diffie–Hellman values originating from an honest partner session. Then, all keys are derived in a way ensuring that (a) from a Diffie–Hellman secret unknown to the adversary sessions derive keys indistinguishable from random (under $\mathsf{PRF\text{-}ODH}$ and PRF assumptions on the $\mathsf{HKDF.Extract}$ and $\mathsf{HKDF.Expand}$ steps) which (b) are independent, allowing revealing and testing of session keys across different stages.

Proof. In the following, we proceed via a sequence of games. Starting from the $\mathsf{Multi\text{-}Stage}$ game, we bound the advantage difference of adversary $\mathcal{A}$ between any two games by complexity-theoretic assumptions until we reach a game where the adversary $\mathcal{A}$ cannot win, i.e., its advantage is at most 0.
Game 0. This is the original $\mathsf{Multi\text{-}Stage}$ game, i.e.,

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} = \mathsf{Adv}^{G_0}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}}.$$

Game 1. In a first step, we restrict the adversary $\mathcal{A}$ in the $\mathsf{Multi\text{-}Stage}$ game to make only a single Test query. That is we can formally turn any multi-query adversary $\mathcal{A}$ into an adversary $\mathcal{A}_1$ which makes only a single Test query. This reduces its advantage, based on a careful hybrid argument, by a factor at most $1/6n_s$ for the six stages in each of the n_s sessions. Note that in the hybrid argument $\mathcal{A}_1$ randomly guesses one of the sessions in advance and only performs the single Test query for this session. The other Test queries of a multi-query attacker are gradually substituted by carefully crafted Reveal queries, where the single-query attacker $\mathcal{A}_1$ needs to know the correct partnering of sessions via session identifiers sid for a correct simulation, e.g., to avoid losses due to bad Reveal-Test combinations on session partners due to the new Reveal queries. The session identifiers sid_1 and sid_2 only contain public information such that partnering is easy to check for them. But then handshake encryption is turned on such that $\mathsf{sid}_3, \ldots, \mathsf{sid}_6$ are based on encrypted data. Fortunately, if the single Test query concerns a (client or server) handshake traffic secret then partnering is easy to decide based on sid_1 resp. sid_2. If the Test query refers to a later key we can reveal the handshake traffic keys of earlier stages, use them to decrypt the subsequent communication, and hence determine $\mathsf{sid}_3, \ldots, \mathsf{sid}_6$ as well. We provide the full details of this hybrid argument in Appendix A.

Incorporating the transformation of $\mathcal{A}$ into $\mathcal{A}_1$ into the game, i.e., by having the challenger guess the right session and making the adaptations, we get

$$\mathsf{Adv}^{G_0}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} \leq 6n_s \cdot \mathsf{Adv}^{G_1}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}}.$$

From now on, we can refer to *the* session label tested at stage i, and we can assume that we know this session number (according to the order of initiated sessions) at the outset of the experiment.
Game 2. In this game, the challenger aborts if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H. We can break the collision resistance of H in case of this event by letting a reduction $\mathcal{B}_1$ output the two distinct input values to H. Thus:

$$\mathsf{Adv}^{G_1}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_2}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

From here on, our security analysis separately considers the two (disjoint) cases that

A. the tested session label has no honest contributive partner in the first stage (i.e., there exists no $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$), and
B. the tested session label has an honest contributive partner in the first stage (i.e., there exists label' with $\mathsf{label.cid}_1 = \mathsf{label}'.\mathsf{cid}_1$).

This allows us to consider the adversary's advantage separately for these two cases A (denoted "test w/o partner") and B ("test w/ partner"):

$$\mathsf{Adv}^{G_2}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_2,\text{ test w/o partner}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} + \mathsf{Adv}^{G_2,\text{ test w/ partner}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}}.$$

Case A. Test without Partner

We first consider the case that the tested session label has no stage-1 contributive partner, which implies it does not have a contributive partner in any stage. By definition, an adversary cannot win if the Test query issued to such session is in a stage that, at the time of the test query, has an unauthenticated peer. Here, authentication refers to the *rectified* level, because the Test oracle checks against this refined property. Hence, for a tested client session, Test (for any stage) cannot be issued before stage 3 is reached and later only if corruption of the client or the partnered server has not taken place before stage 3. Else the adversary loses the game. For a server session, Test can only be issued when stage 6 is reached and client authentication is performed. Here, again, the client cannot be corrupted earlier, else the rectified authentication level would be unauthenticated.

Game A.0. Equals G_2 with adversary restricted to test a session without honest contributive partner in the first stage.

$$\mathsf{Adv}^{G_2,\text{ test w/o partner}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} = \mathsf{Adv}^{G_{A.0}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}}.$$

Game A.1. In this game, we let the challenger guess the peer identity $U \in \mathcal{U}$ of the tested session label (observe that one must be set in order for Test to be admissible, as discussed above), and abort if that guess was incorrect (i.e., $\mathsf{label.pid} \neq U$). This can reduce $\mathcal{A}$'s advantage by a factor at most the number of users n_u:

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} \leq n_u \cdot \mathsf{Adv}^{G_{A.1}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}}.$$

Game A.2. We now let the challenger abort the game if the tested session label receives, within the `CertificateVerify` message from its peer $\mathsf{label.pid} = U$, a valid signature on some (hash value of a) message that has not been computed by any honest session of user U. Note that this message must include the transcript data `ClientHello`$\|\ldots\|$`ClientCert` resp. `ClientHello`$\|\ldots\|$`ServerCert` (cf. Table 2). Observe that, as discussed above, when the Test query is issued to label, such a message must have been received, in the case of a client, prior to accepting stage 3 and with no previous corruption of the server; or, in the case of a server, prior to stage 6 when the server is talking to an authenticating client which is not corrupted yet.

We can bound the probability of Game $G_{A.2}$ aborting for this reason by the advantage of an adversary $\mathcal{B}_2$ against the $\mathsf{EUF\text{-}CMA}$ security of the signature scheme SIG. In the reduction $\mathcal{B}_2$ receives a public key pk_U of a signature scheme, computes the long-term keys of all parties $U' \in \mathcal{U} \setminus \{U\}$ except U and simulates $G_{A.1}$ for $\mathcal{A}_1$. Whenever in that simulation $\mathcal{B}_2$ has to compute a signature under sk_U, it does so via its signing oracle. When label receives a valid signature σ on the (hash value of the) message m, adversary $\mathcal{B}_2$ outputs $(\mathsf{H}(m), \sigma)$ as its forgery. Note that at this point the partnered session cannot be corrupted such that the signature forger does not need to reveal the secret signing key before outputting the forgery.

It remains to argue that the pair $(\mathsf{H}(m), \sigma)$ constitutes a successful forgery. To see this note that the tested session label computes the hash value $\mathsf{H}(m)$ of the message m to verify correctness, but such that no other honest session has computed a signature for this message. According to Game G_2, this also means that no other honest session has derived the same hash value $\mathsf{H}(m') = \mathsf{H}(m)$ for some other message m'. We conclude that the hash value $\mathsf{H}(m)$ has not been signed by user U before.

$$\mathsf{Adv}^{G_{A.1}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} \le \mathsf{Adv}^{G_{A.1}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG},\mathcal{B}_2}$$

It follows for Case A that the adversary cannot make a legitimate Test query at all, unless it forges signatures. Either the sessions do not have a contributive partner, or the sessions in later stages have rejected because of invalid signatures. If the adversary cannot test any session without a contributive partner, it clearly has no advantage in predicting the secret challenge bit b:

$$\mathsf{Adv}^{G_{A.2}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} = 0.$$

Case B. Test with Partner
Game B.0. This is G_2 where the adversary is restricted to issuing a Test query to a session with an honest contributive partner in the first stage.

$$\mathsf{Adv}^{G_2,\text{ test w/ partner}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} = \mathsf{Adv}^{G_{B.0}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}}.$$

Game B.1. In this game, we guess a session $\mathsf{label}' \neq \mathsf{label}$ (from at most n_s sessions in the game) and abort the game if $\mathsf{label.cid}_1 \neq \mathsf{label}'.\mathsf{cid}_1$, i.e., that label' is not the honest contributive partner in stage 1 of the tested session. (Recall that we assume such partner exists in this proof case.) This reduces the adversary's advantage by a factor of at most $1/n_s$.

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}} \le n_s \cdot \mathsf{Adv}^{G_{B.1}}_{\texttt{TLS1.3-full-1RTT},\mathcal{A}}.$$

Game B.2. In this game, we replace the handshake secret HS derived in the tested session and its contributive partner session with a uniformly random and independent string $\widetilde{\mathrm{HS}} \leftarrow_{\$} \{0,1\}^\lambda$. We employ the $\mathsf{dual\text{-}snPRF\text{-}ODH}$ assumption (Definition 2.3) in order to be able to simulate the computation of HS in a partnered client session for a modified `ServerKeyShare` message. More precisely, we can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_3$ against the $\mathsf{dual\text{-}snPRF\text{-}ODH}$ security of the $\mathsf{HKDF.Extract}$ function (taking dES as first and DHE as second input). For this $\mathcal{B}_3$ asks for a PRF challenge on dES computed in the test session and its honest contributive partner. It uses the obtained Diffie–Hellman shares g^x, g^y within `ClientKeyShare` and `ServerKeyShare` of the tested and contributive sessions, and the PRF challenge value as HS in the tested session. If necessary, $\mathcal{B}_3$ uses its $\mathsf{PRF\text{-}ODH}$ queries to derive HS in the partnered session on differing $g^{y'} \neq g^{y}$. Providing a sound simulation of either $G_{B.1}$ (if the bit sampled by the $\mathsf{dual\text{-}snPRF\text{-}ODH}$ challenger was 0 and thus $\widetilde{\mathrm{HS}} = \mathsf{HKDF.Extract}(\mathrm{dES}, g^{xy})$), or $G_{B.2}$ (if the bit sampled by the $\mathsf{dual\text{-}snPRF\text{-}ODH}$ challenger was 1 and thus $\widetilde{\mathrm{HS}} \leftarrow_{\$} \{0,1\}^{\lambda}$), this bounds the advantage difference of $\mathcal{A}$ as:

$$\mathsf{Adv}^{G_{B.1}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.2}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_3}.$$

Game B.3. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the value $\widetilde{\mathrm{HS}}$ replaced in $G_{B.2}$. This affects the derivation of the client handshake traffic secret CHTS, the server handshake traffic secret SHTS and the derived handshake secret dHS in the target session and its matching partner, and the derived handshake secret dHS in all sessions using the same handshake secret $\widetilde{\mathrm{HS}}$. Note that for CHTS and SHTS, these values are distinct from any other session using the same handshake secret value $\widetilde{\mathrm{HS}}$, as the evaluation also takes as input the hash value $H_2 = \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$, (where $\mathtt{CH}$ and $\mathtt{SH}$ contain the client and server random values r_c, r_s respectively) and by Game G_2 we exclude hash collisions. We replace the derivation of CHTS, SHTS and dHS in such sessions with random values $\widetilde{\mathrm{CHTS}}, \widetilde{\mathrm{SHTS}}, \widetilde{\mathrm{dHS}} \leftarrow_{\$} \{0,1\}^{\lambda}$. To ensure consistency, we replace derivations of dHS with the replaced $\widetilde{\mathrm{dHS}}$ sampled by the first session to evaluate $\mathsf{HKDF.Expand}$ using $\widetilde{\mathrm{HS}}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by the previous game, $\widetilde{\mathrm{HS}}$ is a uniformly random value, and the replacement is sound. Thus:

$$\mathsf{Adv}^{G_{B.2}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.3}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_4}.$$

At this point, $\widetilde{\mathrm{CHTS}}$ and $\widetilde{\mathrm{SHTS}}$ are independent of any values computed in any session non-partnered (in stage 1 or 2) with the tested session: distinct session identifiers and no hash collisions (as of Game G_2) ensure that the PRF label inputs for deriving $\widetilde{\mathrm{CHTS}}$ and $\widetilde{\mathrm{SHTS}}$ are unique.

Game B.4. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the values $\widetilde{\mathrm{CHTS}}, \widetilde{\mathrm{SHTS}}$ replaced in $G_{B.3}$. This affects the derivation of the client handshake traffic key $\mathrm{tk_{chs}}$, and the server handshake traffic key $\mathrm{tk_{shs}}$ in the target session and its contributive partner. We replace the derivation of $\mathrm{tk_{chs}}$ and $\mathrm{tk_{shs}}$ with random values $\widetilde{\mathrm{tk_{chs}}} \leftarrow_{\$} \{0,1\}^{L}$ and $\widetilde{\mathrm{tk_{shs}}} \leftarrow_{\$} \{0,1\}^{L}$, where L indicates the sum of key length and iv length for the negotiated AEAD scheme. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of two evaluations of the pseudorandom functions $\mathsf{HKDF.Expand}$. Note that by the previous game $\widetilde{\mathrm{CHTS}}$ and $\widetilde{\mathrm{SHTS}}$ are uniformly random values, and these replacements are sound. Thus:

$$\mathsf{Adv}^{G_{B.3}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.4}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} + 2 \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}.$$

Game B.5. In this game, we replace the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations of the value $\widetilde{\mathrm{dHS}}$ replaced in $G_{B.3}$. This affects the derivation of the master secret MS in any session using the same derived handshake secret $\widetilde{\mathrm{dHS}}$. We replace the derivation of MS in such sessions with the random value $\widetilde{\mathrm{MS}} \leftarrow_{\$} \{0,1\}^{\lambda}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Extract}$. Note that by $G_{B.3}$, $\widetilde{\mathrm{dHS}}$ is a uniformly random value and this replacement is sound. Thus:

$$\mathsf{Adv}^{G_{B.4}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.5}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_6}.$$

Game B.6. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations of the value $\widetilde{\mathrm{MS}}$ replaced in $G_{B.5}$ in the targeted session and its matching session. This affects the derivation of the client application traffic secret CATS, the server application traffic secret SATS, the exporter master secret EMS, and the resumption master secret RMS. For CATS, SATS, and EMS, these evaluations are distinct from any session non-partnered with the tested session, as the evaluation of $\mathsf{HKDF.Expand}$ also takes as input $H_4 = \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SF})$ (where $\mathtt{CH}$ and $\mathtt{SH}$ contain the client and server random values r_c and r_s respectively), and by Game G_2 we exclude hash collisions. For RMS, this evaluation is distinct from any session non-partnered with the tested session, as the evaluation of $\mathsf{HKDF.Expand}$ also takes as input $H_5 = \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CF})$. We replace the derivation of CATS, SATS, EMS, and RMS with random values $\widetilde{\mathrm{CATS}}$, $\widetilde{\mathrm{SATS}}$, $\widetilde{\mathrm{EMS}}$, $\widetilde{\mathrm{RMS}} \leftarrow_{\$} \{0,1\}^{\lambda}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the secret of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by the previous game $\widetilde{\mathrm{MS}}$ is a uniformly random and independent value, and these replacements are sound. Thus:

$$\mathsf{Adv}^{G_{B.5}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.6}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_7}.$$

We note that in this game we have now replaced all stages' keys in the tested session with uniformly random values which, in the protocol execution, are independent of values in any non-partnered session to the tested session. Thus:

$$\mathsf{Adv}^{G_{B.6}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}} = 0.$$

Combining the given single bounds yields the security statement below:

$$\begin{aligned}&\mathsf{Adv}^{G_2,\text{ test w/ partner}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}}\\ &\quad\leq n_s \left(\begin{array}{l}\mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_4} + 2\cdot\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}\\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_7}\end{array}\right)\end{aligned}$$

□

6. Security Analysis of the TLS 1.3 PSK/PSK-(EC)DHE (with Optional 0-RTT) Handshakes

We now turn to analyzing the TLS 1.3 pre-shared key handshakes, with and without Diffie–Hellman key exchange (PSK-(EC)DHE, resp. PSK) and with optional 0-RTT keys, in the pre-shared–secret multi-stage key exchange (sMSKE) model.

Protocol Properties The PSK/PSK-(EC)DHE (0-RTT) handshakes targets the following protocol-specific properties (M, AUTH, FS, USE, REPLAY):

- $\mathsf{M} = 8$: The PSK handshakes with optional 0-RTT consist of eight stages deriving, in order: the early traffic secret ETS and early exporter master secret EEMS (both only in 0-RTT mode), the client and server handshake traffic keys $\mathrm{tk}_{\mathrm{chs}}$ and $\mathrm{tk}_{\mathrm{shs}}$, the client and server application traffic secrets CATS and SATS, the exporter master secret EMS, and the resumption master secret RMS.
- The authentication properties AUTH differ between the PSK(-only) and the PSK-(EC)DHE (0-RTT) handshakes:

  – for PSK (0-RTT), $\mathsf{AUTH} = \big\{((1,1), (2,2), \ldots, (8,8))\big\}$: All keys are immediately mutually authenticated (from the pre-shared key).
  – for PSK-(EC)DHE (0-RTT), $\mathsf{AUTH} = \big\{((1,1), (2,2), (5,8), (5,8), (5,8), (6,8), (7,8), (8,8))\big\}$: The 0-RTT keys ETS/EEMS are always mutually authenticated, the handshake traffic keys $\mathrm{tk}_{\mathrm{chs}}/\mathrm{tk}_{\mathrm{shs}}$ are initially unauthenticated, all non-0-RTT keys reach unilateral authentication with stage 5 and mutual authentication with stage 8. (It is not straightforward to see why some PSK-(EC)DHE keys are not considered to be immediately mutually authenticated, in contrast to keys from the PSK-only handshake. Consider the handshake traffic keys in the PSK-(EC)DHE handshake: in the model, the adversary $\mathcal{A}$ could send its own g^x share to a server session; the server will derive the handshake traffic keys from PSK and DHE. Those keys should now be considered forward secret (due to the ephemeral DH shares); however, when $\mathcal{A}$ corrupts PSK, it can compute the handshake traffic keys. Hence, these keys cannot be treated as forward secret and mutually authenticated at the same time.

- Forward secrecy of the PSK handshake depends on whether an ephemeral Diffie–Hellman key exchange is performed:

  – for PSK-only, $\mathsf{FS} = \infty$: The PSK-only handshake does not provide any forward secrecy.
  – for PSK-(EC)DHE, $\mathsf{FS} = 3$: The PSK-(EC)DHE handshake provides forward secrecy for all non–0-RTT keys.

- $\mathsf{USE} = (\mathsf{internal} : \{3, 4\}, \mathsf{external} : \{1, 2, 5, 6, 7, 8\})$: The handshake traffic keys are used internally to encrypt the second part of the handshake; all other keys are external.
- $\mathsf{REPLAY} = (\mathsf{replayable} : \{1, 2\}, \mathsf{nonreplayable} : \{3, 4, 5, 6, 7, 8\})$: The 0-RTT keys ETS and EEMS are replayable, all other stages' keys are not.

Session and Contributive Identifiers As for the full 1-RTT handshake (cf. Sect. 5), we define the session identifiers over the unencrypted handshake messages; each stage's identifier includes a label and all handshake messages up to when that stage accepts:

$$\begin{aligned}
\mathsf{sid}_1 &= (\text{“ETS”}, && \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}),\\
\mathsf{sid}_2 &= (\text{“EEMS”}, && \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}),\\
\mathsf{sid}_3 &= (\text{“CHTS”}, && \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}, \mathtt{SH}, \mathtt{SKS}^{\dagger}, \mathtt{SPSK}),\\
\mathsf{sid}_4 &= (\text{“SHTS”}, && \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}, \mathtt{SH}, \mathtt{SKS}^{\dagger}, \mathtt{SPSK}),\\
\mathsf{sid}_5 &= (\text{“CATS”}, && \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}, \mathtt{SH}, \mathtt{SKS}^{\dagger}, \mathtt{SPSK}, \mathtt{EE}, \mathtt{SF}),\\
\mathsf{sid}_6 &= (\text{“SATS”}, && \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}, \mathtt{SH}, \mathtt{SKS}^{\dagger}, \mathtt{SPSK}, \mathtt{EE}, \mathtt{SF}),\\
\mathsf{sid}_7 &= (\text{“EMS”}, && \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}, \mathtt{SH}, \mathtt{SKS}^{\dagger}, \mathtt{SPSK}, \mathtt{EE}, \mathtt{SF}),\\
\mathsf{sid}_8 &= (\text{“RMS”}, && \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}, \mathtt{SH}, \mathtt{SKS}^{\dagger}, \mathtt{SPSK}, \mathtt{EE}, \mathtt{SF}, \mathtt{CF}).
\end{aligned}$$

Components indicated with † are present only in the PSK-(EC)DHE variant.

For the contributive identifiers in stages 3 and 4, as for the full handshake we want to ensure server sessions with honest client contribution can be tested, even if the server's response never reaches the client. Therefore, we let client (resp. server) upon sending (resp. receiving) the `ClientHello`, `ClientKeyShare`† and `ClientPreSharedKey` messages set $\mathsf{cid}_3 = (\text{“CHTS”}, \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK})$, $\mathsf{cid}_4 = (\text{“SHTS”}, \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK})$ and later, upon receiving (resp. sending) the `ServerHello`, `ServerKeyShare`† and `ServerPreSharedKey` messages, extend it to $\mathsf{cid}_3 = (\text{“CHTS”}, \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}, \mathtt{SH}, \mathtt{SKS}^{\dagger}, \mathtt{SPSK})$, $\mathsf{cid}_4 = (\text{“SHTS”}, \mathtt{CH}, \mathtt{CKS}^{\dagger}, \mathtt{CPSK}, \mathtt{SH}, \mathtt{SKS}^{\dagger}, \mathtt{SPSK})$. All other contributive identifiers are set to $\mathsf{cid}_i = \mathsf{sid}_i$ (for stages $i \in \{1, 2, 5, 6, 7, 8\}$) when the respective session identifier is set.

6.1. *TLS 1.3 PSK-only (0-RTT optional)*

We can begin to give our security results for the TLS 1.3 PSK-only 0-RTT handshake. We start with Match security.

6.1.1. *Match Security*

Theorem 6.1. *(Match security of* `TLS1.3-PSK-0RTT`*). The TLS 1.3 PSK-only 0-RTT handshake is Match-secure with properties* (M, AUTH, FS, USE, REPLAY) *given above. For any efficient adversary $\mathcal{A}$, there exists an efficient algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{HMAC},\mathcal{B}} + \frac{n_p^2}{|\mathcal{P}|} + n_s^2 \cdot 2^{-|nonce|},$$

where n_s is the maximum number of sessions, n_p is the maximum number of pre-shared secrets, $|\mathcal{P}|$ is the size of the pre-shared secret space, and $|nonce| = 256$ is the bit length of the nonces.

Recall that Match security is a soundness property of the session identifiers. From our definition of session identifiers above, it follows immediately that partnered sessions agree on the derived key, opposite roles, authentication properties, contributive identifiers, and the respective stages. As in the proof of Match security for `TLS1.3-full-1RTT`, the security bound arises as the birthday bound for two honest sessions choosing the same nonce; this not happening ensures at most two partners share the same session identifier.

Proof. We need to show the seven properties of Match security (cf. Definition 4.1).

1. *Sessions with the same session identifier for some stage hold the same key at that stage.*
The session identifiers in each stage include the pre-shared identifier $\mathsf{pssid} = \mathit{pskid}$ (through the `CPSK` and `SPSK` messages, fixing the only key input PSK (as both parties agree upon a mapping $\mathsf{pss}_{U,V}(\mathsf{pssid}) = \mathsf{pss} = \mathrm{PSK}$ to all derived stage keys (recall that $\mathrm{DHE} = 0$ in the TLS 1.3 PSK-only 0-RTT handshake). Furthermore, for each stage, the session identifier includes all handshake messages that enter the key derivation: for stages 1 and 2 messages up to `CPSK` for stages 3 and 4 messages up to `SPSK`, for stages 5, 6, 7 messages up to `SF`, and for stage 8 all messages (up to `CF`). In each stage, the session identifier hence determines *all* inputs to the key derivation, and agreement on it thus ensures agreement on the stage key.
2. *Sessions with the same session identifier for some stage have opposite roles, except for potential multiple responders in replayable stages.*
Assuming at most two sessions share the same session identifier (which we show below), two initiator (client) or responder (server) sessions never hold the same session identifier as they never accept wrong-role incoming messages, and the initial `Hello` messages are typed with the sender's role. This is excluding stages 1 and 2, which are replayable stages in the TLS 1.3 PSK-only 0-RTT handshake.
3. *Sessions with the same session identifier for some stage agree on that stage's authentication level.*
All stages in the TLS 1.3 PSK-only 0-RTT handshake are mutually authenticated, so this is trivially true.
4. *Sessions with the same session identifier for some stage share the same contributive identifier.*
This holds due to, for each stage i, the contributive identifier cid_i being final and equal to sid_i once the session identifier is set.
5. *Sessions are partnered with the intended (authenticated) participant and share the same key identifier.*
All session identifiers include the pssid and binder values sent as part of the `ClientHello`. The pssid thus is trivially agreed upon. The binder value is derived from that PSK through a sequence of $\mathsf{HKDF}/\mathsf{HMAC}$ computations. If we treat HMAC as an unkeyed collision-resistant hash function over both inputs, the key and the message space, agreement on binder implies agreement on PSK. This step is necessary, as $\mathcal{A}$ can set multiple PSK values to share the same pssid, and thus a pssid does not necessarily uniquely determine a pre-shared secret

PSK from each peer's perspective. Instead, we use *binder* to uniquely determine agreement upon PSK between peers. As all PSK values are chosen uniformly at random within the NewSecret query, they collide only with negligible probability, bounded by the birthday bound $n_p^2/|\mathcal{P}|$, where $\mathcal{P}$ is the pre-shared secret space and n_p the maximum number of pre-shared secrets. Therefore, agreement on *binder* and PSK finally implies that pssid, as interpreted by the partnered client and server session, originates from the same NewSecret call. This, from the perspective of both client and server, uniquely identifies the respective peer's identity and hence ensures agreement on the intended peers.

6. *Session identifiers are distinct for different stages.*
This holds trivially as each stage's session identifier has a unique label.

7. *At most two sessions have the same session identifier at any non-replayable stage.*
Recall that stages 1 and 2 are replayable, so we only need to consider stages $i \in \{3, 4, 5, 6, 7, 8\}$. Observe that all session identifiers from these stages include a client and server random nonce (r_c and r_s respectively), through the `ClientHello` and `ServerHello` messages. Therefore, for a threefold collision among session identifiers of honest parties, some session would need to pick the same nonce as one other session (which then may be partnered through a regular protocol run to some third session). The probability for such collision to happen can be bounded from above by the birthday bound $n_s^2 \cdot 2^{-|nonce|}$, where n_s is the maximum number of sessions, and $|nonce| = 256$ the nonces' bit length. □

6.1.2. *Multi-Stage Security*

Theorem 6.2. *(*Multi-Stage *security of* `TLS1.3-PSK-0RTT`*). The TLS 1.3 PSK 0-RTT handshake is* Multi-Stage*-secure with properties* (M, AUTH, FS, USE, REPLAY) *given above. Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1$, ..., $\mathcal{B}_8$ such that*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} \le 8n_s \left(\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_p \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5} \\ + 2 \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_7} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} \end{array} \right) \right)$$

where n_s is the maximum number of sessions, n_u is the maximum number of users, and n_p is the maximum number of pre-shared secrets.

For the TLS 1.3 PSK 0-RTT handshake, Multi-Stage security follows from the security of the pre-shared key: all keys are derived from a pre-shared secret PSK unknown to the adversary (since the PSK mode is not forward secret, PSK may not be corrupted in the tested session) As such, derived keys are indistinguishable from random (under PRF assumptions on the HKDF.Extract and HKDF.Expand steps) which are independent, allowing revealing and testing of session keys across different stages.

Proof. As before, we proceed via a sequence of games, bounding the differences between games via a series of assumptions until we demonstrate that $\mathcal{A}$'s advantage in winning the final game is 0.
Game 0. This is the original $\mathsf{Multi\text{-}Stage}$ game, i.e.,

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} = \mathsf{Adv}^{G_0}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}}.$$

Game 1. We restrict $\mathcal{A}$ to a single Test query, reducing its advantage by a factor of at most $1/8n_s$. Formally, we construct an adversary from $\mathcal{A}$ making only a single Test query via a hybrid argument, analogously to the proof of Theorem 5.2 on page 27, detailed in Appendix A.

$$\mathsf{Adv}^{G_0}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} \leq 8n_s \cdot \mathsf{Adv}^{G_1}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}}.$$

From now on, we can refer to *the* session label tested at stage i, and assume that we know this session in advance.
Game 2. In this game, the challenger aborts if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H. If this event occurs, this can be used to break the collision resistance of H by letting a reduction $\mathcal{B}_1$ (with approximately the same running time as $\mathcal{A}$) output the two distinct input values to H. Thus:

$$\mathsf{Adv}^{G_1}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_2}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Game 3. In this game, the challenger guesses the pre-shared secret PSK used in the tested session, and aborts the game if that guess was incorrect. This reduces $\mathcal{A}$'s advantage by a factor of at most $1/n_p$ for n_p being the maximum number of registered pre-shared secrets, thus:

$$\mathsf{Adv}^{G_2}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} \leq n_p \cdot \mathsf{Adv}^{G_3}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}}.$$

Game 4. In this game, we replace the outputs of the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations using the tested session's guessed pre-shared secret PSK as a key by random values. This affects the derivation of the early secret ES in any session using the same shared PSK. We replace the derivation of ES in such sessions with a random value $\widetilde{\mathrm{ES}} \leftarrow_\$ \{0,1\}^\lambda$. We can bound the difference this step introduces in the advantage of $\mathcal{A}$ by the dual PRF security of $\mathsf{HKDF.Extract}$. Note that any successful adversary cannot issue a $\mathsf{Corrupt}$ query to reveal the PSK used in the tested session, and thus the pre-shared secret is an unknown and uniformly random value, and the simulation is sound. Thus:

$$\mathsf{Adv}^{G_3}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_4}_{\mathtt{TLS1.3\text{-}PSK\text{-}0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2}.$$

Game 5. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the value $\widetilde{\mathrm{ES}}$ replaced in G_4. This affects the derivation of the derived early secret dES, the binder key BK, the early traffic secret ETS, and the early exporter master secret EEMS in any session using the same early secret value $\widetilde{\text{ES}}$ due to the stage being replayable. We replace the derivation of dES, BK, ETS and EEMS in such sessions with random values $\widetilde{\text{dES}}, \widetilde{\text{BK}}, \widetilde{\text{ETS}}, \widetilde{\text{EEMS}} \leftarrow_{\$} \{0, 1\}^{\lambda}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by Game G_4, $\widetilde{\text{ES}}$ is an unknown and uniformly random value, and this replacement is sound. Thus:

$$\mathsf{Adv}^{G_4}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_5}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3}.$$

At this point, we have replaced the stage 1 and stage 2 keys ($\widetilde{\text{ETS}}$ and $\widetilde{\text{EEMS}}$, respectively). We note that if $\mathcal{A}$ issues a $\mathsf{Reveal}(\mathsf{label}, i)$ query to a session label' such that the tested session $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$, then $\mathcal{A}$ would lose the game. Since these stages are replayable, there may be multiple such sessions such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$, however if *any* of these stages is revealed, $\mathcal{A}$ loses the game.

Game 6. In this game, we replace the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations using the value $\widetilde{\text{dES}}$ replaced in G_5. This affects the derivation of the handshake secret HS in any session using the same derived early secret value $\widetilde{\text{dES}}$, as the derivation of HS includes no additional entropy. We replace the derivation of HS in such sessions with a random value $\widetilde{\text{HS}} \leftarrow_{\$} \{0, 1\}^{\lambda}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Extract}$. Note that by the previous game, $\widetilde{\text{dES}}$ is a uniformly random value, and the simulation is sound. Thus:

$$\mathsf{Adv}^{G_5}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_6}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_4}.$$

Game 7. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the value $\widetilde{\text{HS}}$ replaced in G_6. This affects the derivation of the client handshake traffic secret CHTS, the server handshake traffic secret SHTS in the target session and (if it exists) its matching partner, and the derived handshake secret dHS in all sessions using the same handshake secret $\widetilde{\text{HS}}$. Note that for CHTS and SHTS, these values are distinct from any other session using the same handshake secret value $\widetilde{\text{HS}}$, as the evaluation also takes as input the hash value $H_2 = \mathsf{H}(\texttt{CH}\|\texttt{CPSK}\|\texttt{SH}\|\texttt{SPSK})$, where $\texttt{CH}$ and $\texttt{SH}$ contain the client and server random values r_c, r_s respectively, and by Game G_2 we exclude hash collisions. However, dHS may be derived in multiple sessions, as it includes no additional entropy in its computation. We replace the derivation of CHTS, SHTS and dHS in such sessions with random values $\widetilde{\text{CHTS}}, \widetilde{\text{SHTS}}, \widetilde{\text{dHS}} \leftarrow_{\$} \{0, 1\}^{\lambda}$. To ensure consistency, we replace derivations of dHS with the replaced $\widetilde{\text{dHS}}$ sampled by the first session to evaluate $\mathsf{HKDF.Expand}$ using $\widetilde{\text{HS}}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by the previous game, $\widetilde{\text{HS}}$ is a uniformly random value, and the replacement is sound. Thus:

$$\mathsf{Adv}^{G_6}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_7}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}.$$

Game 8. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the values $\widetilde{\text{CHTS}}$, $\widetilde{\text{SHTS}}$ replaced in G_7. This affects the derivation of the client handshake traffic key tk_{chs}, and the server handshake traffic key tk_{shs} in the target session and its matching partner. We replace the derivation of tk_{chs} and tk_{shs} with random values $\widetilde{\text{tk}_{\text{chs}}} \leftarrow_\$ \{0,1\}^L$ and $\widetilde{\text{tk}_{\text{shs}}} \leftarrow_\$ \{0,1\}^L$, where L indicates the sum of key length and iv length for the negotiated AEAD scheme. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of two evaluations of the pseudorandom functions $\mathsf{HKDF.Expand}$. Note that by the previous game $\widetilde{\text{CHTS}}$ and $\widetilde{\text{SHTS}}$ are uniformly random values, and these replacements are sound. Thus:

$$\mathsf{Adv}^{G_7}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_8}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} + 2 \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6}.$$

Game 9. In this game, we replace the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations of the value $\widetilde{\text{dHS}}$ replaced in G_8. This affects the derivation of the master secret MS in any session using the same derived handshake secret dHS. We replace the derivation of MS in such sessions with the random value $\widetilde{\text{MS}} \leftarrow_\$ \{0,1\}^\lambda$. MS may be derived in multiple sessions, as it includes no additional entropy in its computation. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Extract}$. Note that by Game G_7, dHS is a uniformly random value and this replacement is sound. Thus:

$$\mathsf{Adv}^{G_8}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_9}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_7}.$$

Game 10. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations of the value $\widetilde{\text{MS}}$ replaced in G_9 in the targeted session and its matching session. This affects the derivation of the client application traffic secret CATS, the server application traffic secret SATS the exporter master secret EMS and the resumption master secret RMS. For CATS, SATS and EMS, these evaluations are distinct from any other session, as the evaluation of $\mathsf{HKDF.Expand}$ also takes as input $H_4 = \mathsf{H}(\texttt{CH}\|\texttt{CPSK}\|\texttt{SH}\|\texttt{SPSK}\|\texttt{SF})$, where $\texttt{CH}$ and $\texttt{SH}$ contain the client and server random values r_c and r_s respectively, and by Game G_2 we exclude hash collisions. For RMS, this evaluation is distinct from any other session, as the evaluation of $\mathsf{HKDF.Expand}$ also takes as input $H_5 = \mathsf{H}(\texttt{CH}\|\texttt{CPSK}\|\texttt{SH}\|\texttt{SPSK}\|\texttt{SF}\|\texttt{CF})$. We replace the derivation of CATS, SATS, EMS and RMS with random values $\widetilde{\text{CATS}}, \widetilde{\text{SATS}}, \widetilde{\text{EMS}}, \widetilde{\text{RMS}} \leftarrow_\$ \{0,1\}^\lambda$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the secret of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by the previous game $\widetilde{\text{MS}}$ is a uniformly random and independent value, and these replacements are sound. Thus:

$$\mathsf{Adv}^{G_9}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{10}}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}.$$

In Game G_{10} we have now replaced all stages' keys in the tested session with uniformly random values independent from the protocol execution, and thus:

$$\mathsf{Adv}^{G_{10}}_{\texttt{TLS1.3-PSK-0RTT},\mathcal{A}} = 0.$$

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

6.2. *TLS 1.3 PSK-(EC)DHE (0-RTT optional)*

We can now turn to the security results for the TLS 1.3 PSK-(EC)DHE 0-RTT handshake, starting again with Match security.

6.2.1. *Match Security*

Theorem 6.3. *(Match security of* `TLS1.3-PSK-(EC)DHE-0RTT`*). The TLS 1.3 PSK-(EC)DHE 0-RTT handshake is Match-secure with properties* (M, AUTH, FS, USE, REPLAY) *given above. For any efficient adversary $\mathcal{A}$, there exists an efficient algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{HMAC},\mathcal{B}} + \frac{n_p^2}{|\mathcal{P}|} + n_s^2 \cdot \frac{1}{q} \cdot 2^{-|nonce|},$$

where n_s is the maximum number of sessions, q is the group order, n_p is the maximum number of pre-shared secrets, $|\mathcal{P}|$ is the size of the pre-shared secret space, and $|nonce| = 256$ is the bit length of the nonces.

As before, the soundness properties of Match security follow immediately from our definition of session identifiers, with the security bound arising as the birthday bound for two honest sessions choosing the same nonce and group element. The proof hence closely follows the one for Theorem 6.1.

Proof. We need to show the seven properties of Match security (cf. Definition 4.1).

1. *Sessions with the same session identifier for some stage hold the same key at that stage.*
   The session identifiers in each stage include both the pre-shared identifier $pskid$ and the Diffie–Hellman shares g^x and g^y (through the `CPSK` and `SPSK`, and `CKS`, `SKS` messages respectively), fixing both the pre-shared key input PSK and the Diffie–Hellman key input DHE $= g^{xy}$ for all derived stage keys. Furthermore, for each stage, the session identifier includes all handshake messages that enter the key derivation: for stages 1 and 2 messages up to `CPSK`, for stages 3 and 4 messages up to `SPSK`, for stages 5, 6, 7 messages up to `SF`, and for stage 8 all messages (up to `CF`). In each stage, the session identifier hence determines *all* inputs to the key derivation, and agreement on it thus ensures agreement on the stage key.
2. *Sessions with the same session identifier for some stage have opposite roles, except for potential multiple responders in replayable stages.*
   Assuming at most two sessions share the same session identifier (which we show below), two initiator (client) or responder (server) sessions never hold the same session identifier as they never accept wrong-role incoming messages, and the initial `Hello` messages are typed with the sender's role. This is excluding

stages 1 and 2, which are replayable stages in the TLS 1.3 PSK-(EC)DHE 0-RTT handshake.

3. *Sessions with the same session identifier for some stage agree on that stage's authentication level.*
   By definition, the vector determining (upgradable) authentication is fixed to $((1, 1), (2, 2), (5, 8), (5, 8), (5, 8), (6, 8), (7, 8), (8, 8))$, to which hence trivially all sessions agree.
4. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   This holds due to, for each stage i, the contributive identifier cid_i being final and equal to sid_i once the session identifier is set.
5. *Sessions are partnered with the intended (authenticated) participant and share the same key identifier.*
   All session identifiers include the pssid and $binder$ values sent as part of the `ClientHello`. The pssid thus is trivially agreed upon and uniquely determining a pre-shared secret PSK from each peer's perspective. The $binder$ value is derived from that PSK through a sequence of $\mathsf{HKDF}/\mathsf{HMAC}$ computations. If we treat HMAC as an unkeyed collision-resistant hash function over both inputs, the key and the message space, agreement on $binder$ implies agreement on PSK. This step is necessary, as $\mathcal{A}$ can set multiple PSK values to share the same pssid, and thus a pssid does not necessarily uniquely determine a pre-shared secret PSK from each peer's perspective. Instead, we use $binder$ to uniquely determine agreement upon PSK between peers. As all PSK values are chosen uniformly at random within the $\mathsf{NewSecret}$ query, they collide only with negligible probability, bounded by the birthday bound $n_p^2/|\mathcal{P}|$, where $\mathcal{P}$ is the pre-shared secret space and n_p the maximum number of pre-shared secrets. Therefore, agreement on $binder$ and PSK finally implies that pssid, as interpreted by the partnered client and server session, originates from the same $\mathsf{NewSecret}$ call. This, from the perspective of both client and server, uniquely identifies the respective peer's identity and hence ensures agreement on the intended peers.
6. *Session identifiers are distinct for different stages.*
   This holds trivially as each stage's session identifier has a unique label.
7. *At most two sessions have the same session identifier at any non-replayable stage.*
   Recall that stages 1 and 2 are replayable, so we consider only stages $i \in \{3, 4, 5, 6, 7, 8\}$. Recall that all session identifiers from these stages held by some session include a client and server random nonce and Diffie–Hellman share, as all session identifiers contain both the `ClientHello` and `ServerHello` messages. Therefore, for a threefold collision among session identifiers of honest parties, some session would need to pick the same nonce and group element as one other session (which then may be partnered through a regular protocol run to some third session). The probability for such collision to happen can be bounded from above by the birthday bound $n_s^2 \cdot 1/q \cdot 2^{-|nonce|}$, where n_s is the maximum number of sessions, q is the group order, and $|nonce| = 256$ the nonces' bit length. $\square$

6.2.2. *Multi-Stage Security*

Theorem 6.4. *(*$\mathsf{Multi\text{-}Stage}$ *security of* `TLS1.3-PSK-(EC)DHE-0RTT`*). The TLS 1.3 PSK-(EC)DHE 0-RTT handshake is* $\mathsf{Multi\text{-}Stage}$*-secure with properties* (M, AUTH, FS, USE, REPLAY) *given above. Formally, for any efficient adversary* $\mathcal{A}$ *against the* $\mathsf{Multi\text{-}Stage}$ *security there exist efficient algorithms* $\mathcal{B}_1$, ..., $\mathcal{B}_{16}$ *such that*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}}$$

$$\leq 8n_s \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_p n_s \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_7} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_9} \\ + \mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_{10}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{11}} \end{array} \right) \\ \quad + n_s \left(\begin{array}{l} \mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_{12}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{13}} \\ + 2 \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_{15}} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{16}} \end{array} \right) \end{array} \right)$$

where n_s *is the maximum number of sessions,* n_p *the maximum number of pre-shared secrets established between any two parties, and* n_u *is the maximum number of users.*

For the TLS 1.3 PSK-(EC)DHE 0-RTT handshake, $\mathsf{Multi\text{-}Stage}$ security essentially follows from two lines of reasoning. First, the (unforgeable) MAC tags covering (a collision-resistant hash of) the full `Hello` messages ensure that session stages with an authenticated peer share hold exchanged Diffie–Hellman shares originating from an honest partner session. Then, all keys are derived in a way ensuring that (a) for forward-secret stages, the keys are derived from a Diffie–Hellman secret unknown to the adversary are indistinguishable from random (under $\mathsf{PRF\text{-}ODH}$ and $\mathsf{dual\text{-}PRF\text{-}sec/PRF\text{-}sec}$ assumptions on the $\mathsf{HKDF.Extract}$ and $\mathsf{HKDF.Expand}$ steps), and for non-forward-secret stages the keys are derived from a pre-shared secret unknown to the adversary, and are also indistinguishable from random (under PRF assumptions on the $\mathsf{HKDF.Expand}$ and $\mathsf{HKDF.Extract}$ steps) which (b) are independent, allowing revealing and testing of session keys across different stages.

Proof. Again, we proceed via a sequence of games starting from the $\mathsf{Multi\text{-}Stage}$ game and bounding the advantage (differences) of adversary $\mathcal{A}$.
Game 0. This is the original $\mathsf{Multi\text{-}Stage}$ game, i.e.,

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}} = \mathsf{Adv}^{G_0}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}}.$$

Game 1. We again restrict $\mathcal{A}$ to a single Test query, reducing its advantage by a factor of at most $1/8n_s$ via a hybrid argument analogous to the one in the proof of Theorem 5.2 on page 27, detailed in Appendix A.

$$\mathsf{Adv}^{G_0}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq 8n_s \cdot \mathsf{Adv}^{G_1}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}}.$$

From now on, we can refer to *the* session label tested at stage i, and assume to know the session in advance.

Game 2. In this game, the challenger aborts if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H. We can break the collision resistance of H in case of this event by letting a reduction $\mathcal{B}_1$ output the two distinct input values to H. Thus:

$$\mathsf{Adv}^{G_1}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_2}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

From this point, our analysis separately considers the following three (disjoint) cases:

A. that the tested session label has no honest contributive partner in the third stage (i.e., there exists no $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label.cid}_3 = \mathsf{label}'.\mathsf{cid}_3$), and,
B. the tested session label has an honest contributive partner in the third stage (i.e., there exists label' with $\mathsf{label.cid}_3 = \mathsf{label}'.\mathsf{cid}_3$) and $\mathcal{A}$ issues a Test query to the non-forward-secret stages (i.e., $\mathcal{A}$ issues $\mathsf{Test}(\mathsf{label}, i)$ where $i \in \{1, 2\}$.
C. the tested session label has an honest contributive partner in the third stage (i.e., there exists label' with $\mathsf{label.cid}_3 = \mathsf{label}'.\mathsf{cid}_3$) and $\mathcal{A}$ issues a Test query to the forward-secret stages (i.e., $\mathcal{A}$ issues $\mathsf{Test}(\mathsf{label}, i)$ where $i \in \{3, \ldots, 8\}$.

This allows us to consider the adversary's advantage separately for cases A (denoted "test w/o partner"), B (denoted "NFS test w/ partner") and C ("FS test w/ partner"):

$$\begin{aligned}
&\mathsf{Adv}^{G_2}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \\
&\quad\leq \mathsf{Adv}^{G_2,\text{ test w/o partner}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} + \mathsf{Adv}^{G_2,\text{ NFS test w/ partner}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \\
&\quad\quad+ \mathsf{Adv}^{G_2,\text{ FS test w/ partner}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}}.
\end{aligned}$$

Case A. Test without Partner

As before, we first consider the case that the tested session label has no stage 3 contributive partner. For tested initiator sessions, this means that there exists no honest session that has output the received `SH`, `SKS`, and `SPSK` messages. For tested responder session, this means that there exists no honest initiator session that has output the received `CH`, `CKS`, or `CPSK` messages. Since these messages are included in all subsequent stage session identifiers, this implies the tested session does not have a contributive partner in any stage. By definition, an adversary cannot win if the Test query issued to such a session is in a stage that, at the time of the test query, has an unauthenticated peer (where authentication refers to the *rectified* notion). For a tested responder session without an honest contributive partner in stage 3, a Test query can only be issued to the session when it reaches stage 8. For a tested initiator session without an honest contributive partner in stage 3, a Test query can only be issued to the session when it reaches stage 5.

Game A.0. This is identical to Game G_2 with adversary restricted to testing a session without an honest contributive partner in the third stage.

$$\mathsf{Adv}^{G_2,\text{ test w/o partner}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} = \mathsf{Adv}^{G_{A.0}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}}.$$

Game A.1. In this game, the challenger guesses the pre-shared secret PSK used in the tested session, and aborts the game if that guess was incorrect. This reduces $\mathcal{A}$'s advantage by a factor of at most $1/n_p$ (for n_p the maximum number of pre-shared secrets), thus:

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq n_p \cdot \mathsf{Adv}^{G_{A.1}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}}.$$

Game A.2. In this game, the challenger aborts immediately if the initiator (resp. responder) session with label label accepts in the fifth (resp. eighth) stage without an honest contributive partner in stage 3. Let $\mathsf{abort}^{G_{A.2},\mathcal{A}}_{acc}$ denote the event this occurs in $G_{A.2}$. Thus:

$$\left|\mathsf{Adv}^{G_{A.1}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}} - \mathsf{Adv}^{G_{A.2}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}}\right| \leq \Pr[\mathsf{abort}^{G_{A.2},\mathcal{A}}_{acc}]$$

Note that Case A restricts $\mathcal{A}$ to issuing a Test query to a session without an honest contributive partner in stage 3. Because of the authentication type of `TLS1.3-PSK-(EC)DHE-0RTT`, this Test query can only be issued to the initiator (resp. responder) session *after* it reaches stage 5 (resp. 8). Since $G_{A.2}$ is aborted when the session reaches those stages, a successful adversary cannot issue such a query, and thus:

$$\mathsf{Adv}^{G_{A.2}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}} = 0.$$

We now turn to bounding the probability that $\mathsf{abort}^{G_{A.2},\mathcal{A}}_{acc}$ occurs.

Game A.3. In this game, the challenger guesses a session (from at most n_s sessions in the game) and aborts if the guessed session is not the *first* initiator (resp. responder) session which accepts in the fifth (resp. eighth) stage without an honest contributive partner in stage 3. If the challenger guesses correctly (which happens with probability at least $1/n_s$), then this game aborts at exactly the same time as the previous game, and thus:

$$\Pr[\mathsf{abort}^{G_{A.2},\mathcal{A}}_{acc}] \leq n_s \cdot \Pr[\mathsf{abort}^{G_{A.3},\mathcal{A}}_{acc}].$$

We restrict $\mathcal{A}$ from making a $\mathsf{Corrupt}(U, V, k)$ query such that $\mathsf{label.id} = U$, $\mathsf{label.pid} = V$, $\mathsf{label.pssid} = k$, and show that this does not impact $\mathcal{A}$'s advantage in winning this case. By the definition of the case, there does not exist a session label' such that $\mathsf{label}'.\mathsf{cid}_3 = \mathsf{label.cid}_3$ where $\mathcal{A}$'s Test query is issued to label. Since PSK-(EC)DHE mode is unilaterally authenticated in stage 5 and mutually authenticated in stage 8, if the adversary issues a $\mathsf{Corrupt}(U, V, k)$ query before the tested session label (without an honest contributive partner in stage 3) reaches accept in its partner's authenticating stage, when $\mathcal{A}$ issues a $\mathsf{Test}(\mathsf{label}, i)$ query (where $i \in \{1, \ldots, 8\}$) the lost flag is set and $\mathcal{A}$ will lose the game. By the previous games, we abort when the initiator session label (resp. responder session) reaches stage 5 (resp. stage 8) without an honest contributive partner, and thus $\mathcal{A}$ will never issue a $\mathsf{Corrupt}(U, V, k)$ query. In the following games, this will allow us to replace the pre-shared secret pss in the tested session (and all sessions with the same pss value) without being inconsistent or detectable with regard to the $\mathsf{Corrupt}$ query. In what follows, let $\mathsf{pss}_{U,V,k}$ be the guessed pre-shared secret.

Game A.4. In this game, we replace the outputs of the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations using the tested session's guessed pre-shared secret $\mathsf{pss}_{U,V,k}$ as a key by random values. This affects the derivation of the early secret ES in any session using the same shared PSK. We replace the derivation of ES in such sessions with a random value $\widetilde{\mathrm{ES}} \leftarrow_{\$} \{0, 1\}^{\lambda}$. We can bound the difference this step introduces in the advantage of $\mathcal{A}$ by the (dual) security of the pseudorandom function $\mathsf{HKDF.Extract}$. Note that any successful adversary cannot issue a $\mathsf{Corrupt}$ query to reveal $\mathsf{pss}_{U,V,k}$ used in the tested session, and thus the pre-shared secret is an unknown and uniformly random value, and the simulation is sound. Thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.3},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{A.4},\mathcal{A}}] + \mathsf{Adv}_{\mathsf{HKDF.Extract},\mathcal{B}_2}^{\mathsf{dual\text{-}PRF\text{-}sec}}.$$

Game A.5. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the value $\widetilde{\mathrm{ES}}$ replaced in $G_{A.4}$. This affects the derivation of the derived early secret dES, the binder key BK, the early traffic secret ETS, and the early exporter master secret EEMS in any session using the same early secret value $\widetilde{\mathrm{ES}}$ due to the stage being replayable. We replace the derivation of dES, BK, ETS and EEMS in such sessions with random values $\widetilde{\mathrm{dES}}, \widetilde{\mathrm{BK}}, \widetilde{\mathrm{ETS}}, \widetilde{\mathrm{EEMS}} \leftarrow_{\$} \{0, 1\}^{\lambda}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by Game $G_{A.4}$, $\widetilde{\mathrm{ES}}$ is an unknown and uniformly random value, and this replacement is sound. Thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.4},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{A.5},\mathcal{A}}] + \mathsf{Adv}_{\mathsf{HKDF.Expand},\mathcal{B}_3}^{\mathsf{PRF\text{-}sec}}.$$

Game A.6. In this game, we replace the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations using the value $\widetilde{\mathrm{dES}}$ replaced in $G_{A.5}$. This affects the derivation of the handshake secret HS in any session using the same derived early secret value $\widetilde{\mathrm{dES}}$, due to the stage being replayable. We replace the derivation of HS in such sessions with a random value $\widetilde{\mathrm{HS}} \leftarrow_{\$} \{0, 1\}^{\lambda}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Extract}$. Note that by the previous game, $\widetilde{\mathrm{dES}}$ is a uniformly random value, and the simulation is sound. Thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.5},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{A.6},\mathcal{A}}] + \mathsf{Adv}_{\mathsf{HKDF.Extract},\mathcal{B}_4}^{\mathsf{PRF\text{-}sec}}.$$

Game A.7. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the value $\widetilde{\mathrm{HS}}$ replaced in $G_{A.6}$. This affects the derivation of the client handshake traffic secret CHTS, the server handshake traffic secret SHTS in the target session and its matching partner, and the derived handshake secret dHS in all sessions using the same handshake secret $\widetilde{\mathrm{HS}}$. Note that for CHTS and SHTS, these values are distinct from any other session using the same handshake secret value $\widetilde{\mathrm{HS}}$, as the evaluation also takes as input the hash value $H_2 = \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$, (where $\mathtt{CH}$ and $\mathtt{SH}$ contain the client and server random values r_c, r_s respectively) and by Game G_2

we exclude hash collisions. We replace the derivation of CHTS, SHTS and dHS in such sessions with random values $\widetilde{\mathrm{CHTS}}, \widetilde{\mathrm{SHTS}}, \widetilde{\mathrm{dHS}} \leftarrow\!\!\$ \{0,1\}^{\lambda}$. To ensure consistency, we replace derivations of dHS with the replaced $\widetilde{\mathrm{dHS}}$ sampled by the first session to evaluate $\mathsf{HKDF.Expand}$ using $\widetilde{\mathrm{HS}}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by the previous game, $\widetilde{\mathrm{HS}}$ is a uniformly random value, and the replacement is sound. Thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.6},\mathcal{A}}] \le \Pr[\mathsf{abort}_{acc}^{G_{A.7},\mathcal{A}}] + \mathsf{Adv}_{\mathsf{HKDF.Expand},\mathcal{B}_5}^{\mathsf{PRF\text{-}sec}}.$$

Game A.8. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the client handshake traffic secret $\widetilde{\mathrm{CHTS}}$ replaced in $G_{A.7}$. This affects the derivation of the client handshake traffic key $\mathrm{tk}_{\mathrm{chs}}$, and the client finished key fk_C in the target session. We replace the derivation of $\mathrm{tk}_{\mathrm{chs}}$ and fk_C with random values $\widetilde{\mathrm{tk}_{\mathrm{chs}}} \leftarrow\!\!\$ \{0,1\}^L$, $\widetilde{\mathrm{fk}_C} \leftarrow\!\!\$ \{0,1\}^{\lambda}$, where L indicates the sum of key length and iv length for the negotiated AEAD scheme. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by the previous game $\widetilde{\mathrm{CHTS}}$ is a uniformly random value, and these replacements are sound. Thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.7},\mathcal{A}}] \le \Pr[\mathsf{abort}_{acc}^{G_{A.8},\mathcal{A}}] + \mathsf{Adv}_{\mathsf{HKDF.Expand},\mathcal{B}_6}^{\mathsf{PRF\text{-}sec}}.$$

Game A.9. In this game, we show how any adversary that manages to trigger $\mathsf{abort}_{acc}^{G_{A.9},\mathcal{A}}$ (where the tested session has a responder role) can be used to build an adversary $\mathcal{B}_7$ that breaks the existential unforgeability of the HMAC scheme. We let $\mathcal{B}_7$ simulate $G_{A.8}$ for $\mathcal{A}$ as specified, but when the guessed session requires a MAC computation using $\widetilde{\mathrm{fk}_C}$, $\mathcal{B}_7$ instead invokes a MAC oracle to generate that value. Since $\widetilde{\mathrm{fk}_C}$ is a uniformly random and independent value, this simulation is sound. When $\mathcal{A}$ triggers $\mathsf{abort}_{acc}^{G_{A.9},\mathcal{A}}$ (for responder test sessions), the accepting session must have received a `ClientFinished` message that is a valid MAC tag over the hash value $H_4 = \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SF})$. Since all other sessions hold different session identifiers (as there exists no honest contributive partner in the third stage of the accepting session), no honest party will have requested a MAC tag over that session hash. In addition, by Game G_2 there exist no hash collisions, so the MAC input is distinct to all other MAC inputs for any honest party. Thus, this message was never queried to the MAC oracle and is a forgery. This allows us to bound the probability of $\mathcal{A}$ triggering $\mathsf{abort}_{acc}^{G_{A.9},\mathcal{A}}$ due to a stage-8 accepting responder session without a stage-3 contributive partner by:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.8},\mathcal{A}}] \le \Pr[\mathsf{abort}_{acc}^{G_{A.9},\mathcal{A}}] + \mathsf{Adv}_{\mathsf{HMAC},\mathcal{B}_7}^{\mathsf{EUF\text{-}CMA}}$$

Note that for the rest of this case, we bound the probability of an adversary triggering $\mathsf{abort}_{acc}^{G_{A.9},\mathcal{A}}$ when the tested session has an initiator role.

Game A.10. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the server handshake traffic secret $\widetilde{\mathrm{SHTS}}$ replaced in $G_{A.9}$. This affects the derivation of the server handshake traffic key $\mathrm{tk}_{\mathrm{shs}}$, and the server finished key fk_S in the target session. We replace the derivation of $\mathrm{tk}_{\mathrm{shs}}$ and fk_S with random values $\widetilde{\mathrm{tk}_{\mathrm{shs}}}, \widetilde{\mathrm{fk}_S} \leftarrow_\$ \{0,1\}^\lambda$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by a previous game $\widetilde{\mathrm{SHTS}}$ is a uniformly random value, and these replacements are sound. Thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.9},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{A.10},\mathcal{A}}] + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}.$$

Game A.11. In this game, we show how any adversary that manages to trigger $\mathsf{abort}_{acc}^{G_{A.11},\mathcal{A}}$ (where the test session is an initiator session) can be used to build an adversary $\mathcal{B}_9$ that breaks the existential unforgeability of the HMAC scheme. We let $\mathcal{B}_9$ simulate $G_{A.10}$ for $\mathcal{A}$ as specified, but when the guessed session or its partner session requires a MAC computation using $\widetilde{\mathrm{fk}_S}$, $\mathcal{B}_9$ instead invokes a MAC oracle to generate that value. Since $\widetilde{\mathrm{fk}_S}$ is a uniformly random and independent value, this simulation is sound. When $\mathcal{A}$ triggers $\mathsf{abort}_{acc}^{G_{A.11},\mathcal{A}}$ (for initiator test sessions), the accepting session must have received a `ServerFinished` message that is a valid MAC tag over the hash value $H_7 = \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SPSK})$. Since all other sessions hold different session identifiers (as there exists no honest contributive partner in the third stage of the accepting session), no honest party will have requested a MAC tag over that session hash. In addition, by Game G_2 there exist no hash collisions, so the MAC input is distinct to all other MAC inputs for any honest party. Thus, this message was never queried to the MAC oracle and is a forgery. This allows us to bound the probability of $\mathcal{A}$ triggering $\mathsf{abort}_{acc}^{G_{A.11},\mathcal{A}}$ due to a stage-5 accepting initiator session without a stage-3 contributive identifier by:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.11},\mathcal{A}}] \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_9}$$

Combining the given single bounds yield the security statement below:

$$\mathsf{Adv}^{G_2,\text{ test. w/o partner}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq n_p n_s \begin{pmatrix} \mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_4} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_7} \\ + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_9} \end{pmatrix}$$

Case B. NFS Test with Partner

We now turn to the case where the tested session has an honest contributive partner in the third stage, and $\mathcal{A}$ issues a $\mathsf{Test}(\mathsf{label}, i)$ query such that $i \in \{1, 2\}$.

Game B.0. This is identical to Game G_2 with the adversary testing a session with an honest contributive partner in the third stage.

$$\mathsf{Adv}^{G_2,\text{NFS test with partner}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} = \mathsf{Adv}^{G_{B.0}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}}.$$

Game B.1. In this game, we guess the pre-shared secret PSK used in the tested session and abort on a wrong guess. This reduces $\mathcal{A}$'s advantage by a factor of at most $1/n_p$, thus:

$$\mathsf{Adv}^{G_{B.0}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq n_p \cdot \mathsf{Adv}^{G_{B.1}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}}.$$

Game B.2. In this game, we let the challenger guess a session (from at most n_s in the game) and abort if the session guessed is not the honest contributive partner in stage 3 of the tested session. This reduces $\mathcal{A}$'s advantage by a factor of at most $1/n_s$, and thus:

$$\mathsf{Adv}^{G_{B.1}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{B.2}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}}.$$

Game B.3. In this game, we replace the outputs of the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations using the tested session's guessed pre-shared secret $\mathsf{pss}_{U,V,k}$ as a key by random values. This affects the derivation of the early secret ES in any session using the same shared PSK. We replace the derivation of ES in such sessions with a random value $\widetilde{\mathrm{ES}} \leftarrow_\$ \{0,1\}^\lambda$. We can bound the difference this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Extract}$. Note that any successful adversary cannot issue a $\mathsf{Corrupt}$ query to reveal $\mathsf{pss}_{U,V,k}$ used in the tested session (as $\mathcal{A}$ will issue a query $\mathsf{Test}(\mathsf{label}, i)$ such that $i \in \{1,2\}$ by the definition of this case, and $\mathcal{A}$ will cause the lost flag to be set if $\mathsf{Corrupt}(U,V,k)$ is issued), and thus the pre-shared secret is an unknown and uniformly random value, and the simulation is sound. Thus:

$$\begin{aligned}\mathsf{Adv}^{G_{B.2}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq{}& \mathsf{Adv}^{G_{B.3}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}}\\ &+\mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_{10}}.\end{aligned}$$

Game B.4. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations using the value $\widetilde{\mathrm{ES}}$ replaced in $G_{B.3}$. This affects the derivation of the derived early secret dES, the binder key BK, the early traffic secret ETS, and the early exporter master secret EEMS in any session using the same early secret value $\widetilde{\mathrm{ES}}$ due to the stage being replayable. We replace the derivation of dES, BK, ETS and EEMS in such sessions with random values $\widetilde{\mathrm{dES}}, \widetilde{\mathrm{BK}}, \widetilde{\mathrm{ETS}}, \widetilde{\mathrm{EEMS}} \leftarrow_\$ \{0,1\}^\lambda$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by Game $G_{B.3}$, $\widetilde{\mathrm{ES}}$ is an unknown and uniformly random value, and this replacement is sound. Thus:

$$\begin{aligned}\mathsf{Adv}^{G_{B.3}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}} \leq{}& \mathsf{Adv}^{G_{B.4}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RTT},\mathcal{A}}\\ &+\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{11}}.\end{aligned}$$

We note that at this point, we have replaced the stage 1 and stage 2 keys ($\widetilde{\mathrm{ETS}}$ and $\widetilde{\mathrm{EEMS}}$, respectively). We note that if $\mathcal{A}$ issues a $\mathsf{Reveal}(\mathsf{label}, i)$ query to a session label' such that the tested session $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$, then $\mathcal{A}$ would lose the game. Since these stages are replayable, then there may be multiple such sessions such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$. Since $\widetilde{\mathrm{ETS}}$ and $\widetilde{\mathrm{EEMS}}$ are now uniformly random values independent of the protocol execution, we have:

$$\mathsf{Adv}^{G_{B.4}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} = 0.$$

Combining the given single bounds yields the security statement below:

$$\mathsf{Adv}^{G_2,\text{NFS test with partner}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} \leq n_s n_p \big(\mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_{10}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{11}}\big)$$

Case C. FS Test with Partner
We now turn to the third case, "FS Test with Partner", where the tested session has an honest contributive partner in the third stage, and $\mathcal{A}$ issues a $\mathsf{Test}(\mathsf{label}, i)$ query such that $i \in \{3, \ldots, 8\}$.
Game C.0. This is identical to Game G_2 with the adversary testing a session with an honest contributive partner in the third stage.

$$\mathsf{Adv}^{G_2,\text{FS test with partner}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} = \mathsf{Adv}^{G_{C.0}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}}.$$

Game C.1. In this game, we let the challenger guess a session (from at most n_s in the game) and abort if the session guessed is not the honest contributive partner in stage 3 of the tested session. This reduces $\mathcal{A}$'s advantage by a factor of at most $1/n_s$ and thus:

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{C.1}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}}.$$

Game C.2. In this game, we replace the handshake secret HS derived in the tested session and its contributive partner session with a uniformly random and independent string $\widetilde{\text{HS}} \leftarrow_{\$} \{0,1\}^{\lambda}$. We employ the dual-snPRF-ODH assumption in order to be able to simulate the computation of HS in a partnered client session for a modified `ServerKeyShare` message. More precisely, we can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_{12}$ against the dual-snPRF-ODH security of the HKDF.Extract function (taking dES as first and DHE as second input). For this, $\mathcal{B}_{12}$ asks for a PRF challenge on dES. It uses the obtained Diffie-Hellman shares g^x, g^y within `ClientKeyShare` and `ServerKeyShare` of the tested session and its contributive partner session, and the PRF challenge value as HS in the test session. If necessary, $\mathcal{B}_{12}$ uses its PRF-ODH queries to derive HS in the partnered session on differing $g^{y'} \neq g^y$. Providing a sound simulation of either $G_{C.1}$ (if the bit sampled by the dual-snPRF-ODH challenger was 0 and thus $\widetilde{\text{HS}} = \mathsf{PRF}(\text{dES}, g^{xy})$) or $G_{C.2}$ (if the bit sampled by the dual-snPRF-ODH challenger was 1 and thus $\widetilde{\text{HS}} \leftarrow_{\$} \{0,1\}^{\lambda}$), this bounds the advantage difference of $\mathcal{A}$ as:

$$\mathsf{Adv}^{G_{C.1}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.2}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_{12}}.$$

Game C.3. In this game, we replace the pseudorandom function HKDF.Expand in all evaluations using the value $\widetilde{\text{HS}}$ replaced in $G_{C.2}$. This affects the derivation of the client handshake traffic secret CHTS, the server handshake traffic secret SHTS in the target session and its matching partner, and the derived handshake secret dHS in all sessions using the same handshake secret $\widetilde{\text{HS}}$. Note that for CHTS and SHTS, these values are distinct from any other session using the same handshake secret value $\widetilde{\text{HS}}$, as the evaluation also takes as input the hash value $H_2 = \mathsf{H}(\texttt{CH}\|\texttt{SH})$, (where CH and SH contain the client and server random values r_c, r_s respectively) and by Game G_2 we exclude hash collisions. We replace the derivation of CHTS, SHTS and dHS in such sessions with random values $\widetilde{\text{CHTS}}, \widetilde{\text{SHTS}}, \widetilde{\text{dHS}} \leftarrow_\$ \{0,1\}^\lambda$. To ensure consistency, we replace derivations of dHS with the replaced $\widetilde{\text{dHS}}$ sampled by the first session to evaluate HKDF.Expand using $\widetilde{\text{HS}}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function HKDF.Expand. Note that by the previous game, $\widetilde{\text{HS}}$ is a uniformly random value, and the replacement is sound. Thus:

$$\mathsf{Adv}^{G_{C.2}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.3}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{13}}.$$

At this point, $\widetilde{\text{CHTS}}$ and $\widetilde{\text{SHTS}}$ are independent of any values computed in any session non-partnered (in stage 1 or 2) with the tested session: distinct session identifiers and no hash collisions (as of Game G_2) ensure that the PRF label inputs for deriving $\widetilde{\text{CHTS}}$ and $\widetilde{\text{SHTS}}$ are unique.

Game C.4. In this game, we replace the pseudorandom function HKDF.Expand in all evaluations using the values $\widetilde{\text{CHTS}}, \widetilde{\text{SHTS}}$ replaced in $G_{C.3}$. This affects the derivation of the client handshake traffic key tk_{chs}, and the server handshake traffic key tk_{shs} in the target session and its matching partner. In the derivation, we replace tk_{chs} and tk_{shs} with random values $\widetilde{\text{tk}_{\text{chs}}} \leftarrow_\$ \{0,1\}^L$ and $\widetilde{\text{tk}_{\text{shs}}} \leftarrow_\$ \{0,1\}^L$, where L indicates the sum of key length and iv length for the negotiated AEAD scheme. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of two evaluations of the pseudorandom functions HKDF.Expand. Note that by the previous game $\widetilde{\text{CHTS}}$ and $\widetilde{\text{SHTS}}$ are uniformly random values, and these replacements are sound. Thus:

$$\mathsf{Adv}^{G_{C.3}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.4}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RTT},\mathcal{A}} + 2 \cdot \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}}.$$

Game C.5. In this game, we replace the pseudorandom function HKDF.Extract in all evaluations of the value $\widetilde{\text{dHS}}$ replaced in Game $G_{C.4}$. This affects the derivation of the master secret MS in any session using the same derived handshake secret dHS. We replace the derivation of MS in such sessions with the random value $\widetilde{\text{MS}} \leftarrow_\$ \{0,1\}^\lambda$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function HKDF.Extract. Note that by Game $G_{C.3}$, $\widetilde{\text{dHS}}$ is a uniformly random value and this replacement is sound. Thus:

$$\mathsf{Adv}^{G_{C.4}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.5}}_{\texttt{TLS1.3-PSK-(EC)DHE-0RT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_{15}}.$$

Game C.6. In this game, we replace the pseudorandom function $\mathsf{HKDF.Expand}$ in all evaluations of the value $\widetilde{\mathrm{MS}}$ replaced in $G_{C.5}$ in the targeted session and its matching session. This affects the derivation of the client application traffic secret CATS, the server application traffic secret SATS the exporter master secret EMS and the resumption master secret RMS. For CATS, SATS and EMS, these evaluations are distinct from any other session, as the evaluation of $\mathsf{HKDF.Expand}$ also takes as input $H_4 = \mathsf{H}(\mathtt{CH}\|\mathtt{SH}\|\mathtt{SF})$ (where $\mathtt{CH}$ and $\mathtt{SH}$ contain the client and server random values r_c and r_s respectively), and by Game G_2 we exclude hash collisions. For RMS, this evaluation is distinct from any other session, as the evaluation of $\mathsf{HKDF.Expand}$ also takes as input $H_5 = \mathsf{H}(\mathtt{CH}\|\mathtt{SH}\|\mathtt{SF}\|\mathtt{CF})$. We replace the derivation of CATS, SATS, EMS, and RMS with random values $\widetilde{\mathrm{CATS}}$, $\widetilde{\mathrm{SATS}}$, $\widetilde{\mathrm{EMS}}$, $\widetilde{\mathrm{RMS}} \leftarrow_{\$} \{0,1\}^{\lambda}$. We can bound the difference that this step introduces in the advantage of $\mathcal{A}$ by the secret of the pseudorandom function $\mathsf{HKDF.Expand}$. Note that by the previous game $\widetilde{\mathrm{MS}}$ is a uniformly random and independent value, and these replacements are sound. Thus:

$$\mathsf{Adv}^{G_{C.5}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RT},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.6}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RT},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{16}}.$$

We note that in this game we have now replaced all stages' keys (with the restriction that the tested stage is from stages 3-8) in the tested session with uniformly random values independent of the protocol execution and thus

$$\mathsf{Adv}^{G_{C.6}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RT},\mathcal{A}} = 0.$$

Combining the given single bounds yields the security statement below:

$$\mathsf{Adv}^{G_2,\text{FS test with partner}}_{\mathtt{TLS1.3\text{-}PSK\text{-}(EC)DHE\text{-}0RT},\mathcal{A}} \leq n_s \begin{pmatrix} \mathsf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF.Extract},\mathbb{G},\mathcal{B}_{12}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{13}} \\ +2\cdot\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_{15}} \\ +\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{16}} \end{pmatrix}$$

□

7. Discussion and Conclusions

Our analysis provides several insights into the design and properties of the TLS 1.3 handshake and indicates potential avenues for future research.

7.1. *Technical Differences from Our Earlier Work*

As noted in the introduction, this paper is a successor to earlier versions of our work [38,39,47,53,61]. Here we briefly comment on the technical differences of the analyses of `draft-05` in [38], `draft-10` and `draft-dh` in [39], and `draft-14` in [53], compared to the final version of TLS 1.3 analyzed in this paper. We focus on three main aspects: the stages identified for the multi-stage analysis, the session identifiers of those stages, and the assumptions used in the security proofs. For the stages and session identifiers, the changes across our series of works are directly related to how the protocol flows and key schedule evolved.

Stages—Main Handshake `draft-05-(EC)DHE` had 3 stages: handshake traffic key, application traffic key, and the resumption master secret RMS. `draft-dh` and `draft-10-(EC)DHE` added the exporter master secret EMS. In this paper we have 6 stages capturing the final RFC's main handshake: handshake traffic keys $\mathrm{tk}_{\mathrm{chs}}$ and $\mathrm{tk}_{\mathrm{shs}}$; application traffic secrets CATS and SATS; and EMS and RMS. The main reason this paper has 2 stages for the handshake traffic keys and 2 stages for the application traffic secrets is a change to the key schedule: the earlier drafts had 4 secrets (client write key, client write IV, server write key, server write IV) derived from each of two secrets (handshake traffic key and application traffic key), whereas TLS 1.3 has 2 secrets (write key, write IV) derived from each of four secrets ($\mathrm{tk}_{\mathrm{chs}}$, $\mathrm{tk}_{\mathrm{shs}}$, CATS, SATS).

Stages—PSK Handshake `draft-05-SR` had 2 stages: handshake traffic key and application traffic key. `draft-10-PSK` added EMS. `draft-14-PSK-0RTT` added an early handshake secret and an early application data secret. In this paper we have 8 stages capturing the final RFC's PSK handshake: early traffic secret ETS; early exporter master secret EEMS; handshake traffic keys $\mathrm{tk}_{\mathrm{chs}}$ and $\mathrm{tk}_{\mathrm{shs}}$; application traffic secrets CATS and SATS; and EMS and RMS. Again the main reason for the additional stages in this paper is the aforementioned change to the key schedule.

Session Identifiers In the main handshake, session identifiers for the handshake traffic keys are the same across [38,39] and this paper. For the application keys, session identifiers changed based on changes in the message flow which caused changes to the transcript included in the session hash used for key derivation. In particular, `draft-05-(EC)DHE` and `draft-10-(EC)DHE` included `ClientCertificate` in the application key session identifiers but not `ServerFinished`, whereas TLS 1.3 analyzed in this paper does not include `CCRT` but does include `SF`. Similarly, session identifiers for the PSK handshakes changed across the papers due to changes in message ordering and what messages were available to be included in the session hash.

Cryptographic Assumptions—Main Handshake The cryptographic assumptions used in the proofs for `draft-05-(EC)DHE`, `draft-dh`, `draft-10-(EC)DHE`, and this paper remain the same. (In early papers we used the notation $\mathsf{PRF\text{-}ODH}$ rather than the newer notation $\mathsf{snPRF\text{-}ODH}$ introduced by [17], but the actual assumption was the same.)

Cryptographic Assumptions—PSK Handshake In `draft-05`, no (EC)DHE variant of the PSK handshake was present (then called "session resumption handshake"), so the proof of `draft-05-SR` relied solely on symmetric-key assumptions. `draft-10-PSK` relied on the same assumptions as `draft-05-SR`, whereas `draft-10-PSK-(EC)DHE` added an $\mathsf{EUF\text{-}CMA}$ assumption on HMAC as well as the $\mathsf{PRF\text{-}ODH}$ assumption. `draft-14-PSK-0RTT` and `draft-14-PSK-(EC)DHE-0RTT` added a randomness assumption on HMAC, which in the analysis of the final RFC's `TLS1.3-PSK-0RTT` and `TLS1.3-PSK-(EC)DHE-0RTT` in this paper is superseded by a dual-PRF-sec assumption on HKDF.Extract in the multi-stage security bounds. The latter more explicitly indicates those places where HKDF.Extract is keyed through the second argument, which were treated more implicitly in the theorem statements of earlier versions.

Match-security of `TLS1.3-PSK-0RTT` and `TLS1.3-PSK-(EC)DHE-0RTT` in this paper adds a collision resistance assumption on HMAC due to the introduction of the PSK binder.

7.2. *Comments on the TLS 1.3 Design*

Value of Key Separation Earlier versions of TLS used 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 [25] as noted in [56,63,86], which motivated the combined handshake+record layer analysis in the authenticated and confidential channel establishment model of [64]. We confirm that the change in keys for encryption of handshake messages allows keys established during the TLS 1.3 handshake to achieve standard key indistinguishability security.

Key Independence All forms of the TLS 1.3 handshake achieve key independence for all stage keys: one can reveal one stage's session key without endangering the security of later-stage keys. This follows from the fact that every key exported or used for encryption is a leaf node in the directed graph representing the key schedule in Fig. 2. Beyond making it amenable to generic composition, key independence safeguards the usage of derived keys against inter-protocol effects of security breakdowns. (Some early drafts had less key independence: for example, in `draft-05`, each exported key was derived directly from the master secret MS. Since MS was also used to derive other keys, it could not be considered as an output stage key, so every exported key had to be included directly in the main analysis. Contrast this with the final approach in which an exporter master secret EMS is derived from MS, and then all exported keys are derived from EMS: we can treat EMS as an output stage key, and consider the derivation of exported keys as a symmetric protocol using EMS that is composed with the TLS 1.3 handshake protocol.)

A "Dent" in the Key Schedule In terms of key derivation, we remark that there is a noteworthy "dent" in the TLS 1.3 key schedule (cf. Fig. 2): all second-level secrets derived from the main (early/handshake/master) secrets are used solely to derive traffic encryption keys (in case of traffic secrets) or further purposes (resumption and exporting), *except* for the handshake traffic secrets CHTS/SHTS which, beyond deriving the handshake traffic keys, are also used to compute the finished keys. This allowed us to define all but the handshake traffic secrets as output session keys in the multi-stage key exchange sense, while requiring to descend one level further to capture the handshake traffic keys.

A more uniform key schedule could have derived the finished keys in a separate branch from the handshake secret HS, enabling CHTS/SHTS to become first-order session keys on the same level as all others. This in turn would allow a more uniform interface for composition with arbitrary symmetric-key protocol and possibly better support the treatment of key updates (cf. [60]). While this is only a minor issue for the TLS 1.3 analysis, it turned out to complicate a modular analysis of the TLS 1.3 handshake integration into the QUIC protocol [94] as remarked by Delignat-Lavaud et al. [41].

Including the Session Hash in Signatures and Key Derivation In the TLS 1.3 full handshake, authenticating parties (the server, and sometimes the client) sign (the hash of) all handshake messages up to when the signature is issued (the "session hash"). This is different from TLS 1.2 and earlier, where the server's signature is only over the client and server random nonces and the server's ephemeral public key.

As for key derivation, every stage key is derived using a PRF application that includes the hash of all messages exchange up to the point when the stage key is derived.

In our analysis, the session identifier for each stage is set to be the transcript of messages up to that point. Thus, assuming collision resistance of the hash function, different session identifiers result in different keys. (This was the goal of the session hash which was introduced in response to the triple handshake attack [13] on TLS 1.2 and earlier.) The server signing the transcript also facilitates our proofs of the authentication properties in the full handshake.

Furthermore, if output 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` [8] and proposed `tls-unique-prf` [65] 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 the PSK handshake without (EC)DHE, 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 pre-shared key as the key. All sessions sharing the same pre-shared secret then compute the same master secret. However, since derivation of output keys still uses the session hash as context, output keys are unique assuming uniqueness of protocol messages (which is assured for example by unique nonces).

Encryption of Handshake Messages A major design goal of TLS 1.3 was to enhance privacy (against passive adversaries) by encrypting the second part of the handshake (which contains identity certificates) using the initial handshake traffic keys $\mathrm{tk}_{\mathrm{chs}}$ and $\mathrm{tk}_{\mathrm{shs}}$. Our analysis shows that the handshake traffic keys do indeed have security against passive adversaries (and even active adversaries by the time the client handshake traffic key $\mathrm{tk}_{\mathrm{chs}}$ is used) and hence this feature of TLS 1.3 does increase the handshake's privacy. The secrecy of the remaining stage keys however do not rely on the handshake being encrypted and would remain secure even if the handshake was done in clear.

Finished Messages The `Finished` messages sent by both client and server at the end of the TLS 1.3 handshake are MAC values computed by applying HMAC to the (hash of the) handshake transcript, keyed by dedicated client/server finished secrets $\mathrm{fk}_C/\mathrm{fk}_S$.

Interestingly, according to our proofs, the `Finished` messages do not contribute to the implicit authentication and secrecy of the output keys in the full handshake or the PSK-only handshake, in the sense that the key exchange would achieve the same security notion without these messages. This is mainly because, in the full handshake, the signatures already authenticate the transcripts, and, in the PSK-only handshake, all keys are derived from the PSK which provides implicit authentication. While `Finished` messages are not needed to provide implicit authentication in PSK-only handshakes, they would play a role in providing explicit authentication, but our model does not include an explicit authentication property. In the PSK-(EC)DHE handshake, the `Finished` messages do contribute authentication of the ephemeral Diffie–Hellman public keys under (a key derived from) the PSK. The `Finished` messages can still generally be interpreted as providing some form of (explicit) session key confirmation and authentication [42,55,61].

Compare these with the case of RSA key transport in the TLS 1.2 full handshake: the analyses of both Krawczyk et al. [70] and Bhargavan et al. [19] note potential weaknesses or require stronger security assumptions if `Finished` messages are omitted.

Upstream Hashing in Signatures, MACs, and Key Derivation In signing (resp. MACing) the transcript for authentication as well as in deriving keys via HKDF, TLS 1.3 uses the *hash* of the current transcript as input; if, e.g., 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 respective signature, MAC, or KDF algorithms opaquely take care of processing this message. For engineering purposes, however, it may be desirable to hash the transcript iteratively, only storing the intermediate values instead of the entire transcript. In our security proof, this upstream hashing introduces the collision-resistant assumption for the hash function (and hence a potential additional source of weaknesses, cf. [15]), which would otherwise be taken care of by the signature, MAC, resp. KDF scheme.

0-RTT Replays and Forward Secrecy Through our analysis, we capture the effects of replays in the cryptographic security sense, most importantly confirming that the replayability of 0-RTT keys has no negative effects on the cryptographic security of subsequently derived keys. From a practical, application-layer perspective, the potential for 0-RTT replays, however, remains a critical design choice in TLS 1.3 and has been subject of controversial discussion (see, e.g., [83]). The TLS 1.3 standard [90, Sect. 8] acknowledges that "TLS does not provide inherent replay protections for 0-RTT data," and at the same time urges implementations to at least implement a certain basic level of anti-replay protection (like single-use session tickets, `ClientHello` recording, or freshness checks). The 0-RTT modes of Google's QUIC protocol and TLS 1.3 spawned a series of academic treatments of 0-RTT key exchange [52,62,78] and new designs of forward-secure encryption [30,59] to achieve forward-secret and non-replayable 0-RTT key exchange [46,57] and TLS session resumption [6].

Also, from a cryptographic perspective, the Diffie–Hellman-based 0-RTT mode variant offered a higher level of (forward) security as it did not require the client to keep secret state for resumption and hence only server compromises would affect the secrecy of 0-RTT communication [53,74]. This handshake variant was abandoned with `draft-13` in favor of performance and structural simplification.

7.3. *Open Research Questions*

Composition Key exchange protocols would be of limited use if applied in isolation; generally, the derived keys are meant to be deployed in a follow-up (or overall) protocol. Encryption (and authentication) of application data via a (cryptographic) channel protocol is of course a common approach, with the TLS record protocol being a prime example, but other usage in the TLS setting includes exporting of key material or resumption handshakes (via the exporter resp. resumption master secret).

Key exchange protocols secure in the sense of Bellare–Rogaway [25] are indeed amenable to generic secure composition with arbitrary follow-up symmetric protocols as shown by Brzuska et al. [22,26]. Earlier versions of our work [38,61] included adaptations of these composition results to the multi-stage setting, demonstrating that stage keys could be safely used in symmetric key protocols. Those results still apply to our current model, when restricted to stage keys that are marked for external use, are non-replayable, and when treating the authentication characteristic as fixed at acceptance time, not upgradable. However, it is not obvious how to translate the notions of upgradable authentication or replayability generically to a symmetric key protocol. Given that our focus is on the TLS 1.3 handshake protocol as an authenticated key exchange protocol, we leave a composition result translating replayability and upgradable authentication to future work.

As part of a composed treatment of the overall TLS 1.3 protocol (i.e., handshake and record layer), a conceptual alternative to our treatment of the handshake could be to consider *all* keys—including handshake traffic keys—to be external (from the handshake's perspective), and rely on the record protocol for handshake encryption. This viewpoint is taken especially in analyses based on verified implementations [40] and would, in the computational setting, require an appropriate amalgamation of channel models capturing the bidirectional, multi-key, multiplexed, and streaming nature of the TLS 1.3 record protocol [23,54,60,85,88].

Post-Quantum Key Exchange While our theorems are mostly generic in terms of cryptographic assumptions, they do directly rely on a Diffie–Hellman assumption in a group. Post-quantum key exchange, however, is usually formulated generically as a key encapsulation mechanism (KEM). If TLS 1.3 is to be extended to support post-quantum or hybrid (i.e., traditional plus post-quantum) key exchange [36], our results on the full 1-RTT and PSK-(EC)DHE modes will need to be revisited in the context of specific post-quantum KEMs or generic properties of KEMs. As we rely on the $\mathsf{PRF\text{-}ODH}$ assumption [17], an interactive assumption which provides some notion of "active security", it may be the case that translating our proofs to the KEM setting requires use of an IND-CCA KEM. Brendel et al. [16] discuss challenges arising when moving Diffie–Hellman-style key exchanges to the post-quantum setting and Schwabe et al. [93] present a KEM-based alternative to the TLS 1.3 handshake with modified message flow.

7.4. *Conclusions*

In this work, we have updated our prior analyses of the cryptographic security of several draft TLS 1.3 handshakes to the final, standardized version of TLS 1.3 in RFC 8446 [90]. We analyzed the full 1-RTT handshake mode as well as the PSK-based resumption handshake modes, with optional 0-RTT keys, in the reductionist framework of an enhanced multi-stage key exchange security model that captures the various security properties of the several keys derived in TLS 1.3. Our analysis confirmed that the TLS 1.3 handshake follows sound cryptographic design principles and establishes session keys with their desired security properties under standard cryptographic assumptions.

The IETF TLS working group developed TLS 1.3 through a novel, proactively transparent standardization process (cf. [89]) that actively solicited industry and academia alike. In our opinion, this has led to an unprecedented success in having wide-ranging security analyses for a major Internet security protocol *prior* to its standardization and deployment. While security models or formal method tools can never capture the entirety of real-world threats to such protocols, we believe that, through this process, the boundaries of formal understanding have been pushed to an extent that significantly strengthens confidence in the soundness of TLS 1.3's design. As such, the TLS 1.3 standardization process exemplifies a commendable paradigm which rightfully is being adopted for standardization processes of other major Internet security protocols, and which we encourage other standards bodies to adopt.

Acknowledgements

We thank Markulf Kohlweiss for insightful discussions on the necessity of the PRF-ODH assumption for proofs of the TLS 1.3 handshakes. We thank Håkon Jacobsen for comments on the proof for the pre-shared key handshake, Joseph Jaeger and Damiano Abram for comments on the formalization of Match security, and Denis Diemert for comments on the handshake protocol figures. We also thank the reviewers of this and earlier versions of this work for valuable comments. Benjamin Dowling was supported by EPSRC grant EP/L018543/1. The work of Marc Fischlin has been funded in part by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity. Felix Günther has been supported in part by Research Fellowship grant GU 1859/1-1 of the German Research Foundation (DFG) and National Science Foundation (NSF) grants CNS-1526801 and CNS-1717640. Douglas Stebila was supported by Australian Research Council (ARC) Discovery Project grant DP130104304, Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grant RGPIN-2016-05146, and NSERC Discovery Accelerator Supplement grant RGPIN-2016-05146. This work has been co-funded by the DFG as part of project S4 within the CRC 1119 CROSSING.

A Reducing Multiple to Single Test Queries

In this section we give more details on the hybrid argument to reduce adversaries $\mathcal{A}_{\text{multi}}$ which make multiple Test queries in the $\mathsf{Multi\text{-}Stage}$ game for the TLS 1.3 handshakes to adversaries $\mathcal{A}_{\text{single}}$ which restrict themselves to a single Test query. Note that any multi-query adversary cannot make more (reasonable) Test queries than the number n_s of overall sessions times the maximum number M of stages. Any adversary making more queries needs to repeat queries for some keys, yielding the reply $\perp$, and such queries can be easily sorted out.

The main step is the hybrid argument where adversary $\mathcal{A}_{\text{single}}$ simulates $\mathcal{A}_{\text{multi}}$'s attack, making only a single Test query. To do so, for a randomly chosen index n between 1 and the maximum number n_s of Test queries, adversary $\mathcal{A}_{\text{single}}$ returns the genuine keys in the first $n-1$ queries of $\mathcal{A}_{\text{multi}}$, poses the Test query for the n-th query as its own query, and returns random keys from query $n+1$ on. To get the genuine keys for the first queries, $\mathcal{A}_{\text{single}}$ instead calls the Reveal oracle.

The above works along the common argument in hybrid games if we can ensure that $\mathcal{A}_{\text{single}}$ does not lose because of the additional Reveal queries it makes, i.e., if it reveals a key for the partner of the (only) Test session. One option to ensure this is to demand that $\mathcal{A}_{\text{multi}}$ never tests a session and its partner. Luckily, the multi-stage security model in Sect. 4 supports this smoothly. Namely, testing a session in stage i for which the tested_i flag has been already set to true will immediately return $\perp$. This setting of the flag happens in one of the following cases:

- When the session itself is tested for the first time in this stage, or
- if the session accepts at this stage after a Send call and there is already a partner with $\mathsf{tested}_i = \mathsf{true}$ (triggered in the Send execution for which the session accepts), or
- if it is partnered to a tested session and has just accepted at the same stage (triggered through the testing of the partner in the Test oracle execution).

Furthermore, a Test call to a session stage i for which the session has already passed, i.e., $\mathsf{st_{exec}} \neq \mathsf{accepted}_i$, also returns $\perp$. In other words, any Test query of $\mathcal{A}_{\text{multi}}$ to a partner of a previous Test query returns $\perp$. We can therefore avoid such queries and let $\mathcal{A}_{\text{single}}$ answer $\perp$ for such queries directly. In this case the remaining Reveal queries, substituting the first Test queries in the hybrid argument, cannot cause $\mathcal{A}_{\text{single}}$ to lose, because now they are for sure not partnered with the (only) Test query of $\mathcal{A}_{\text{single}}$.

There are two caveats in the above reasoning. First, adversary $\mathcal{A}_{\text{single}}$ needs to know if two session stages are partnered in order to correctly respond $\perp$ for some Test queries. While this is trivial to deduce from the public communication data for sid_1 and sid_2, the session identifiers $\mathsf{sid}_3, \ldots, \mathsf{sid}_6$ contain confidentially transmitted messages, protected through the handshake traffic keys derived in stages 1 and 2 in the TLS 1.3 handshake. But if we let $\mathcal{A}_{\text{single}}$ know these two $\mathsf{internal}$ keys via carefully selected Reveal queries when testing for a stage $i \geq 3$ then it can decrypt the communication and decide partnering with other sessions for this stage. Since these further Reveal queries are for earlier stages, they essentially cannot interfere with the stage of the Test session.

It is convenient to store the information about tested sessions in an internal array $\mathsf{simTested}_i[\mathsf{label}]$ which is set to true if session label would have been marked as tested_i in the game, if $\mathcal{A}_{\text{multi}}$ would have actually made that query. We write this as an array in order to distinguish this internal list to $\mathcal{A}_{\text{single}}$ from entries in sessions label. At any point in time, the array $\mathsf{simTested}$ in $\mathcal{A}_{\text{single}}$'s simulation will hold the same information as the entries tested if $\mathcal{A}_{\text{multi}}$ had actually made all Test queries.

The second issue arises from the fact that internal keys (i.e., keys in stages i with $\mathsf{USE}_i = \mathsf{internal}$) are overwritten in partners to tested sessions. This can happen in the $\mathsf{Send}(\mathsf{label}, m)$ command if a partner label of a tested session stage for label' (with $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$) goes to $\mathsf{accepted}_i$. Then the security game sets $\mathsf{label}.\mathsf{key}_i \leftarrow \mathsf{label}'.\mathsf{key}_i$. The other case can occur in the $\mathsf{Test}(\mathsf{label}, i)$ query itself if a partnered session label' to the tested session is already in state $\mathsf{label}'.\mathsf{st_{exec}} = \mathsf{accepted}_i$. Then the internal key of

that session label' is replaced by the answer for the tested session. But our adversary $\mathcal{A}_{\text{single}}$ with a single Test query of course only sets one session stage to be tested, influencing at most one further session, whereas $\mathcal{A}_{\text{multi}}$'s multiple Test queries may overwrite several keys.
Since there is no other mechanism to modify keys in the security model, we need to take care of the issue manually in the simulation of $\mathcal{A}_{\text{multi}}$ through $\mathcal{A}_{\text{single}}$. Fortunately, the internal keys in TLS 1.3 handshake are only used for protecting the data in transport, wrapping and unwrapping the data immediately when sending or receiving. We thus let $\mathcal{A}_{\text{single}}$ keep internal arrays $\mathsf{actualKey}_i[\mathsf{label}]$ and $\mathsf{simKey}_i[\mathsf{label}]$ for the internal keys (in the actual attack of $\mathcal{A}_{\text{single}}$, resp. in the simulation of $\mathcal{A}_{\text{multi}}$) at stage $i \in \{1, 2\}$ and let $\mathcal{A}_{\text{single}}$ adapt authenticated encryptions with respect to such keys when relaying them between the simulation $\mathcal{A}_{\text{multi}}$ and Send.

Lemma A.1. *Let $\mathcal{A}_{\text{multi}}$ be an adversary making at most $n_{\mathsf{Test}} \leq \mathsf{M} \cdot n_s$ calls to* Test *attacking TLS 1.3 full 1-RTT handshake in the* $\mathsf{Multi\text{-}Stage}$ *game. Then, there exists an adversary $\mathcal{A}_{\text{single}}$ which makes only a single* Test *query such that*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}_{\text{multi}}} \leq n_{\mathsf{Test}} \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}_{\text{single}}}.$$

In addition, $\mathcal{A}_{\text{single}}$ initiates the same maximum number n_s of sessions as $\mathcal{A}_{\text{multi}}$.

The lemma holds analogously for the other handshake variants of TLS 1.3 since the argument uses only specifics which are shared by all variants.

Proof. We build our adversary $\mathcal{A}_{\text{single}}$ from $\mathcal{A}_{\text{multi}}$ via a black-box simulation. Adversary $\mathcal{A}_{\text{single}}$ proceeds as follows. Initially, it picks $n \in \{1, 2, \ldots, n_{\mathsf{Test}}\}$ at random and initializes empty arrays $\mathsf{actualKey}_i[\,] \leftarrow \bot$, $\mathsf{simKey}_i[\,] \leftarrow \bot$ for $i = 1, 2$ and $\mathsf{simTested}_i[\,] \leftarrow \mathsf{false}$ for $i \in \{1, 2, \ldots, \mathsf{M}\}$. Algorithm $\mathcal{A}_{\text{single}}$ then invokes $\mathcal{A}_{\text{multi}}$, relaying all oracle queries except for the Send and Test queries.

Simulating Send queries. A $\mathsf{Send}(\mathsf{label}, m)$ query is answered by possibly switching encryptions, if the session has been marked as a (virtually) tested session such that keys need to be adapted. Let $i = \mathsf{label.stage}$ denote the current stage, meaning that the session has already accepted at stage i:

- If $i \leq 1$ or $m = \mathsf{init}$ or $m = \mathsf{continue}$ then pass the command to the own Send oracle. Such messages are not encrypted, and we do not need to re-encrypt the communication data.
- If $i \geq 2$ and $\mathsf{simTested}_i[\mathsf{label}] = \mathsf{true}$, i.e., the data is encrypted under a key which has potentially changed due to the (virtual) test, then re-encrypt with the client resp. server handshake traffic key in the experiment of $\mathcal{A}_{\text{single}}$. Note that session identifiers (esp. for stages $i \geq 3$) are not affected by this re-encryption as they are defined over the cleartexts:

  – If $\mathsf{label.role} = \mathsf{initiator}$, i.e., we expect the client to receive a message protected under the server's traffic handshake secret (the stage-2 key), then decrypt m with key $\mathsf{simKey}_2[\mathsf{label}]$ and re-encrypt the result with $\mathsf{actualKey}_2[\mathsf{label}]$ to m' before passing (label, m') to the own Send oracle. Here, and in the following, we assume that encryption always succeeds for messages different from $\bot$, and that $\bot$ is encrypted to something which again decrypts to $\bot$.
  – If $\mathsf{label.role} = \mathsf{responder}$, i.e., we expect the server to receive a message protected under the client's traffic handshake secret (the stage-1 key), then decrypt m with key $\mathsf{simKey}_1[\mathsf{label}]$ and re-encrypt the result with $\mathsf{actualKey}_1[\mathsf{label}]$ to m' before passing (label, m') to the own Send oracle.

- In any other case just forward (label, m) to the own Send oracle.

For the response m from the Send oracle do the following:

- If $i \leq 1$ then hand back the response unchanged.
- If $i \geq 2$ and $\mathsf{simTested}_i[\mathsf{label}] = \mathsf{true}$, then adapt encryption to the keys expected by $\mathcal{A}_{\text{multi}}$:

  · If $\mathsf{label.role} = \mathsf{initiator}$ then decrypt m with key $\mathsf{actualKey}_1[\mathsf{label}]$ and re-encrypt the result with $\mathsf{simKey}_1[\mathsf{label}]$ to m' before returning m'.

- · If $\mathsf{label.role} = \mathsf{responder}$ then decrypt m with key $\mathsf{actualKey}_2[\mathsf{label}]$ and re-encrypt the result with $\mathsf{simKey}_2[\mathsf{label}]$ to m' before returning m'.

- In any other case return m.

In addition, check if one needs to set the status of simTested. If the Send call changes the status to $\mathsf{accepted}_{i+1}$ —about which the adversary $\mathcal{A}_{\text{single}}$ is informed— then do the following:

- For $i + 1 \leq 2$, if there is a session $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label}'.\mathsf{sid}_{i+1} = \mathsf{label}.\mathsf{sid}_{i+1}$ and $\mathsf{simTested}_{i+1}[\mathsf{label}'] = \mathsf{true}$, then set the test status for the session here, $\mathsf{simTested}_{i+1}[\mathsf{label}] \leftarrow \mathsf{true}$. Note that since the session identifiers in the first two stages consists of the cleartext messages, this is easy to check in this case. Also copy the internal keys, $\mathsf{actualKey}_{i+1}[\mathsf{label}] \leftarrow \mathsf{actualKey}_{i+1}[\mathsf{label}']$ and $\mathsf{simKey}_{i+1}[\mathsf{label}] \leftarrow \mathsf{simKey}_{i+1}[\mathsf{label}']$.
- For $i + 1 \geq 3$, if $\mathsf{simTested}_1[\mathsf{label}] = \mathsf{true}$ then fetch the key $\mathsf{actualKey}_1[\mathsf{label}]$, else make a Reveal query (label, 1), and analogously for the key for stage 2. Since the session under consideration has already accepted in stage $i + 1 \geq 3$ at this point, our adversary $\mathcal{A}_{\text{single}}$ obtains the two handshake traffic keys and uses these keys to decrypt the communication (in its attack) to recover sid_{i+1} in session label. Compare this value to the session identifiers in all sessions label' with $\mathsf{simTested}_{i+1}[\mathsf{label}'] = \mathsf{true}$ for the same stage $i + 1$. Note that a session label' can only be partnered in stage $i + 1 \geq 3$ if it is already partnered in the first two stages, because sid_{i+1} contains the identifiers sid_1 and sid_2 as prefix (except for the label). This also implies that such sessions can only derive the same handshake traffic keys. Thus, we can use the same handshake keys as for label to decrypt for label'. If there is a match, then update $\mathsf{simTested}_{i+1}[\mathsf{label}] \leftarrow \mathsf{true}$ and copy the keys from session label' as before, $\mathsf{actualKey}_{i+1}[\mathsf{label}] \leftarrow \mathsf{actualKey}_{i+1}[\mathsf{label}']$ and $\mathsf{simKey}_{i+1}[\mathsf{label}] \leftarrow \mathsf{simKey}_{i+1}[\mathsf{label}']$.

Except for the copying of the actual key, this now corresponds exactly to the update step in the Send query.

Simulating Test queries. The t-th Test query (label, i) is answered as follows:

- If there is no session label, or the session label has not accepted in stage i yet—which is known to the adversary because it gets to learn $\mathsf{label.st_{exec}}$ upon successful completion of the i-th stage— or $\mathsf{simTested}_i[\mathsf{label}] = \mathsf{true}$, then immediately return $\perp$.
- Otherwise proceed as follows:

  – If $t < n$ then make a Reveal(label, i) call to get the key K and return it to $\mathcal{A}_{\text{multi}}$. Set $\mathsf{simTested}_i[\mathsf{label}] \leftarrow \mathsf{true}$ and, if $i \leq 2$ and the key is internal, also set $\mathsf{actualKey}_i[\mathsf{label}] \leftarrow K$ and $\mathsf{simKey}_i[\mathsf{label}] \leftarrow K$.
  – If $t = n$ then make a Test(label, i) call and return the answer K to $\mathcal{A}_{\text{multi}}$. Set $\mathsf{simTested}_i[\mathsf{label}] \leftarrow \mathsf{true}$ and, if $i \leq 2$, also set $\mathsf{actualKey}_i[\mathsf{label}] \leftarrow K$ and $\mathsf{simKey}_i[\mathsf{label}] \leftarrow K$.
  – If $t > n$ then pick a key $K \leftarrow\$ \mathcal{D}$ randomly and return it to $\mathcal{A}_{\text{multi}}$. Set $\mathsf{simTested}_i[\mathsf{label}] \leftarrow \mathsf{true}$ and, if $i \leq 2$, this time define $\mathsf{actualKey}_i[\mathsf{label}] \leftarrow \mathsf{Reveal}(\mathsf{label}, i)$ and $\mathsf{simKey}_i[\mathsf{label}] \leftarrow K$.

- Finally, we need to check as in the original Test query if there is already a partnered session in accepted state for the same stage, and, if so, modify its status. If there exists a session $\mathsf{label}' \neq \mathsf{label}$ which is partnered, $\mathsf{label}'.\mathsf{sid}_i = \mathsf{label}.\mathsf{sid}_i$, and where $\mathsf{label}'.\mathsf{st_{exec}} = \mathsf{label}.\mathsf{st_{exec}} = \mathsf{accepted}_i$, then set $\mathsf{simTested}_i[\mathsf{label}'] \leftarrow \mathsf{true}$. If $i \leq 2$ then also copy the keys, $\mathsf{actualKey}_i[\mathsf{label}'] \leftarrow \mathsf{actualKey}_i[\mathsf{label}]$ and $\mathsf{simKey}_i[\mathsf{label}'] \leftarrow \mathsf{simKey}_i[\mathsf{label}]$. We note that the checking against a match to label' is done analogously to the Send query.

We remark that we do not alter the simulated Reveal oracle but let queries through without modifications. There are cases now where the multi-query adversary $\mathcal{A}_{\text{multi}}$ may thus obtain a different internal key than expected. But this can only happen for sessions which have been (virtually) tested, such that any Reveal query or such a partnered session where the key has been changed would make $\mathcal{A}_{\text{multi}}$ lose. We thus ignore these cases and simply continue with the misaligned answer.

Analysis Note that the additional Reveal queries, which $\mathcal{A}_{\text{single}}$ makes for internal keys in the simulation of Send and Test above, cannot interfere with its only Test query. Recall that $\mathcal{A}_{\text{single}}$ may make Reveal(label, 1) and Reveal(label, 2) queries when simulating the Send and Test queries.
In the simulated Send query we need to check that the potential Reveal(label, 1) and Reveal(label, 2) queries for the internal keys in stages 1 and 2 do not conflict with the (only) Test query for session $\mathsf{label_{tested}}$

which $\mathcal{A}_{\text{single}}$ makes for stage i. Note that these queries would only be made if the session label has accepted at a stage ≥ 3. Assume that indeed $i \in \{1, 2\}$ and that $\mathsf{label}_{\mathsf{tested}}$ is partnered with label in that stage i. For this distinguish the point in time when the Test call to $\mathsf{label}_{\mathsf{tested}}$ is made:

- If the Test call for $\mathsf{label}_{\mathsf{tested}}$ is made later, after session label has continued after accepting in stage $i \leq 2$, then the adversary $\mathcal{A}_{\text{multi}}$ loses. The reason is that in this case there is a partnered session label which has continued beyond stage $i \in \{1, 2\}$ and has used the $\mathsf{internal}$ key already. Such a Test call sets $\mathsf{lost} \leftarrow \mathsf{true}$ according to the model.
- If the Test call for $\mathsf{label}_{\mathsf{tested}}$ has already been made for stage $i \in \{1, 2\}$ before session label has continued after accepting in that stage, then the session (label, i) partnered to $(\mathsf{label}_{\mathsf{tested}}, i)$ for $i \in \{1, 2\}$ must have been marked as tested, $\mathsf{simTested}_i[\mathsf{label}] = \mathsf{true}$, in a simulation of Test or Send (without using Reveal queries for this stage with cleartext session identifiers). In this case, however, our algorithm $\mathcal{A}_{\text{single}}$ does not make a Reveal query for this stage but reads off the key from the array $\mathsf{actualKey}_i[\mathsf{label}]$.

Hence, in the first case we can only increase the success probability and in the second case we avoid a conflicting Reveal query straight away. Note that the same is true for the final check in the simulation of Test. It remains to argue the compatibility of the other potential Reveal queries in the simulated Test query. If the n-th query for session $\mathsf{label}_{\mathsf{tested}}$ and stage i (which $\mathcal{A}_{\text{single}}$ forwards to its Test oracle) would be partnered with the t-th query (label, i), then the call of $\mathcal{A}_{\text{multi}}$ to its (simulated) Test oracle for $\mathsf{label}_{\mathsf{tested}}$ later

- would either make $\mathcal{A}_{\text{multi}}$ lose if the session $\mathsf{label}_{\mathsf{tested}}$ was at the point of the query already past the state $\mathsf{accepted}_i$ for the $\mathsf{internal}$ key (according to the description of the Test oracle), or
- the session $\mathsf{label}_{\mathsf{tested}}$ is already in state $\mathsf{accepted}_i$ when the test query here is made, in which case the (simulated) Test oracle would mark that session $\mathsf{label}_{\mathsf{tested}}$ as tested, $\mathsf{simTested}_i[\mathsf{label}_{\mathsf{tested}}] = \mathsf{true}$, because it is partnered to the now (virtually) tested session label, or
- the session $\mathsf{label}_{\mathsf{tested}}$ is not yet in state $\mathsf{accepted}_i$ when the test query here is made, in which case later the (simulated) Send oracle would mark that session $\mathsf{label}_{\mathsf{tested}}$ as tested when it eventually accepts in stage i, $\mathsf{simTested}_i[\mathsf{label}_{\mathsf{tested}}] = \mathsf{true}$, because it is then partnered to the (virtually) tested session label here.

We ignore the first case because it cannot contribute to $\mathcal{A}_{\text{multi}}$'s success probability. For the latter two cases it follows that the n-th query of $\mathcal{A}_{\text{multi}}$ will actually not be forwarded to $\mathcal{A}_{\text{single}}$'s oracle Test, because for such marked sessions with $\mathsf{simTested}_i[\mathsf{label}_{\mathsf{tested}}] = \mathsf{true}$ the simulated Test oracle immediately returns $\perp$. Hence, $\mathcal{A}_{\text{single}}$ does not make any Test query in these cases at all, and in particular cannot lose because of a Reveal query for a session partnered to the one in the Test query.
By the above it follows that $\mathcal{A}_{\text{single}}$ only sets lost in its attack if $\mathcal{A}_{\text{multi}}$ does so in the simulation. For the final step in the analysis of the hybrid argument observe that if $n = 1$ and $b_{\mathsf{test}} = 0$ (for the challenge bit in $\mathcal{A}_{\text{single}}$'s game) then our adversary $\mathcal{A}_{\text{single}}$ only returns random keys to $\mathcal{A}'_{\text{multi}}$ (or error messages $\perp$) in simulated Test queries. Furthermore, unless $\mathcal{A}'_{\text{multi}}$ loses the game, the simulation is perfectly sound in the sense that it has the same distribution as in an actual attack; in particular this argument is not violated by the re-encryption. Hence, in this case we have that $\mathcal{A}_{\text{single}}$ predicts its value b_{test} with the same probability as $\mathcal{A}'_{\text{multi}}$ when receiving only random keys in all (valid) Test queries. Analogously, if $n = n_{\mathsf{Test}}$ and $b_{\mathsf{test}} = 1$ then $\mathcal{A}_{\text{single}}$ always returns genuine keys (or errors) to $\mathcal{A}'_{\text{multi}}$, again, in a sound simulation unless $\mathcal{A}'_{\text{multi}}$ loses. This therefore corresponds to the case that $\mathcal{A}'_{\text{multi}}$ only receives genuine keys in all (valid) Test queries.
For the analysis, let b be the output of $\mathcal{A}_{\text{single}}$ and b_{test} be its challenge bit. Similarly, let b' be the output of $\mathcal{A}_{\text{multi}}$ in an actual attack for test bit b'_{test}. We denote by $b = b_{\mathsf{test}}$ resp. $b' = b'_{\mathsf{test}}$ the events that the bit is correct and the lost flag is not set. Then,

$$\begin{aligned}
\Pr\left[b = b_{\mathsf{test}}\right] &= \sum_{n_0=1}^{n_{\mathsf{Test}}} \Pr\left[b = b_{\mathsf{test}} \wedge n = n_0\right] \\
&= \frac{1}{n_{\mathsf{Test}}} \cdot \sum_{n_0=1}^{n_{\mathsf{Test}}} \Pr\left[b = b_{\mathsf{test}} \mid n = n_0\right] \\
&= \frac{1}{n_{\mathsf{Test}}} \cdot \sum_{n_0=1}^{n_{\mathsf{Test}}} \Big(\tfrac{1}{2} \cdot \Pr\left[b = 0 \mid b_{\mathsf{test}} = 0 \wedge n = n_0\right]
\end{aligned}$$

$$
\begin{aligned}
&+ \tfrac{1}{2} \cdot \left(1 - \Pr\left[b = 0 \mid b_{\mathsf{test}} = 1 \wedge n = n_0\right]\right)\Big) \\
&= \tfrac{1}{2} + \tfrac{1}{n_{\mathsf{Test}}} \cdot \sum_{n_0=1}^{n_{\mathsf{Test}}} \tfrac{1}{2} \cdot \Big(\Pr\left[b = 0 \mid b_{\mathsf{test}} = 0 \wedge n = n_0\right] \\
&\quad - \Pr\left[b = 0 \mid b_{\mathsf{test}} = 1 \wedge n = n_0\right]\Big)
\end{aligned}
$$

and noting that the simulation conditioned on $b_{\mathsf{test}} = 1$ and $n = n_0$ is equivalent to the simulation for $b_{\mathsf{test}} = 0$ and $n = n_0 + 1$, the telescope sum simplifies to

$$
\begin{aligned}
&= \tfrac{1}{2} + \tfrac{1}{n_{\mathsf{Test}}} \cdot \tfrac{1}{2}\Big(\Pr\left[b = 0 \mid b_{\mathsf{test}} = 0 \wedge n = 1\right] - \Pr\left[b = 0 \mid b_{\mathsf{test}} = 1 \wedge n = n_{\mathsf{Test}}\right]\Big) \\
&= \tfrac{1}{2} + \tfrac{1}{n_{\mathsf{Test}}} \cdot \tfrac{1}{2}\Big(\Pr\left[b = 0 \mid b_{\mathsf{test}} = 0 \wedge n = 1\right] - 1 + \Pr\left[b = 1 \mid b_{\mathsf{test}} = 1 \wedge n = n_{\mathsf{Test}}\right]\Big) \\
&\geq \tfrac{1}{2} + \tfrac{1}{n_{\mathsf{Test}}} \cdot \Big(\Pr\left[b' = b'_{\mathsf{test}}\right] - \tfrac{1}{2}\Big)
\end{aligned}
$$

where we used in the last step that the simulation is perfectly sound (if $\mathcal{A}_{\text{multi}}$ does not lose) and thus at least the probability in an actual attack. We remark that our adversary $\mathcal{A}_{\text{single}}$ may trigger $\mathsf{lost} \leftarrow \mathsf{true}$ less often than $\mathcal{A}_{\text{multi}}$, e.g., because of the omitted Test queries and potential conflicts with Reveal queries. Hence, we obtain

$$
\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}'_{\text{multi}}} \leq n_{\mathsf{Test}} \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathtt{TLS1.3\text{-}full\text{-}1RTT},\mathcal{A}_{\text{single}}},
$$

proving the claim of the lemma. □

References

[1] L. Akhmetzyanova, E. Alekseev, E. Smyshlyaeva, A. Sokolov, Continuing to reflect on TLS 1.3 with external PSK. Cryptology ePrint Archive, Report 2019/421 (2019). https://eprint.iacr.org/2019/421

[2] 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, P. Zimmermann, Imperfect forward secrecy: How Diffie-Hellman fails in practice, in *ACM CCS 15* (2015)

[3] G. Arfaoui, X. Bultel, P.-A. Fouque, A. Nedelcu, C. Onete, The privacy of the TLS 1.3 protocol. *PoPETs*, **2019**(4), 190–210 (2019). https://doi.org/10.2478/popets-2019-0065

[4] N. AlFardan, D.J. Bernstein, K.G. Paterson, B. Poettering, J.C.N. Schuldt, On the security of RC4 in TLS, in *Proceedings of 22nd USENIX Security Symposium*, pp. 305–320. USENIX (2013)

[5] M. Abdalla, M. Bellare, P. Rogaway, The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pp. 143–158. Springer, Heidelberg, (2001). https://doi.org/10.1007/3-540-45353-9_12

[6] N. Aviram, K. Gellert, T. Jager, Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. *Journal of Cryptology* (2021). To appear. Available as Cryptology ePrint Archive, Report 2019/228. https://eprint.iacr.org/2019/228

[7] N.J. AlFardan, K.G. Paterson, Lucky thirteen: breaking the TLS and DTLS record protocols, in *2013 IEEE Symposium on Security and Privacy*, pp. 526–540. IEEE Computer Society Press, (2013). https://doi.org/10.1109/SP.2013.42

[8] J. Altman, N. Williams, L. Zhu, Channel Bindings for TLS. RFC 5929 (Proposed Standard), 2010. http://www.ietf.org/rfc/rfc5929.txt

[9] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, J.K. Zinzindohoue, A messy state of the union: taming the composite state machines of TLS, in *2015 IEEE Symposium on Security and Privacy*, pp. 535–552. IEEE Computer Society Press, 2015. https://doi.org/10.1109/SP.2015.39

[10] B. Beurdouche, K. Bhargavan, A. Delignat-Levaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, J.K. Zinzindohoue, A messy state of the union: Taming the composite state machines of TLS, in *Proceedings of IEEE Symposium on Security & Privacy (S&P) 2015*, pp. 535–552. IEEE (2015)

[11] K. Bhargavan, C. Brzuska, C. Fournet, M. Green, M. Kohlweiss, S. Zanella-Béguelin. Downgrade resilience in key-exchange protocols, in *2016 IEEE Symposium on Security and Privacy*, pp. 506–525. IEEE Computer Society Press, 2016. https://doi.org/10.1109/SP.2016.37

[12] M. Bellare, R. Canetti, H. Krawczyk, Keying hash functions for message authentication, in Neal Koblitz, editor, *CRYPTO'96, vol. 1109 of LNCS*, pp. 1–15. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_1

[13] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, P.-Y. Strub, Triple handshakes and cookie cutters: breaking and fixing authentication over TLS, in *2014 IEEE Symposium on Security and Privacy*, pp. 98–113. IEEE Computer Society Press (2014). https://doi.org/10.1109/SP.2014.14

[14] M. Bellare, New proofs for NMAC and HMAC: Security without collision-resistance, in Cynthia Dwork, editor, *CRYPTO 2006, vol. 4117 of LNCS*, pp. 602–619. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_36

[15] J. Brendel, M. Fischlin, F. Günther, Breakdown resilience of key exchange protocols: NewHope, TLS 1.3, and hybrids, in K. Sako, S. Schneider, P.Y.A. Ryan, editors, *ESORICS 2019, Part II*, volume 11736 of *LNCS*, pp. 521–541. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29962-0_25

[16] J. Brendel, M. Fischlin, F. Günther, C. Janson, D. Stebila, Challenges in proving post-quantum key exchanges based on key encapsulation mechanisms. Cryptology ePrint Archive, Report 2019/1356, 2019. https://eprint.iacr.org/2019/1356

[17] J. Brendel, M. Fischlin, F. Günther, C. Janson, PRF-ODH: relations, instantiations, and impossibility results. In J. Katz, H. Shacham, editors, *CRYPTO 2017, Part III, vol. 10403 of LNCS*, pp. 651–681. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-63697-9_22

[18] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, Implementing TLS with verified cryptographic security, in *2013 IEEE Symposium on Security and Privacy*, pp. 445–459. IEEE Computer Society Press (2013). https://doi.org/10.1109/SP.2013.37

[19] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, S. Z. Béguelin, Proving the TLS handshake secure (as it is). In J.A. Garay, R. Gennaro, editors, *CRYPTO 2014, Part II, vol. 8617 of LNCS*, pp. 235–255. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_14

[20] K. Bhargavan, C. Fournet, M. Kohlweiss, miTLS: verifying protocol implementations against real-world attacks. *IEEE Security & Privacy*, **14**(6), 18–25 (2016) https://doi.org/10.1109/MSP.2016.123

[21] C. Brzuska, M. Fischlin, N.P. Smart, B. Warinschi, S.C. Williams. Less is more: Relaxed yet composable security notions for key exchange. *International Journal of Information Security*, **12**(4), 267–297 (2013) https://doi.org/10.1007/s10207-013-0192-y

[22] C. Brzuska, M. Fischlin, B. Warinschi, S.C. Williams, Composability of Bellare-Rogaway key exchange protocols, in Y. Chen, G. Danezis, V. Shmatikov, editors, *ACM CCS 2011*, pp. 51–62. ACM Press (2011). https://doi.org/10.1145/2046707.2046716

[23] C. Boyd, B. Hale, Secure channels and termination: the last word on TLS. In T. Lange, O. Dunkelman, editors, *LATINCRYPT 2017, vol. 11368 of LNCS*, pp. 44–65. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-030-25283-0_3

[24] C. Badertscher, C. Matt, U. Maurer, P. Rogaway, B. Tackmann, Augmented secure channels and the goal of the TLS 1.3 record layer. In M. H. Au, A. Miyaji, editors, *ProvSec 2015, vol. 9451 of LNCS*, pp. 85–104. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-26059-4_5

[25] M. Bellare, P. Rogaway, Entity authentication and key distribution. In D.R. Stinson, editor, *CRYPTO'93, vol. 773 of LNCS*, pp. 232–249. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_21

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

[27] M. Bellare, B. Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3, in M. Robshaw, J. Katz, editors, *CRYPTO 2016, Part I, vol. 9814 of LNCS*, pp. 247–276. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_10

[28] K. Cohn-Gordon, C. Cremers, K. Gjøsteen, H. Jacobsen, T. Jager, Highly efficient key exchange protocols with optimal tightness, in A. Boldyreva, D. Micciancio, editors, *CRYPTO 2019, Part III, vol. 11694 of LNCS*, pp. 767–797. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-26954-8_25

[29] C. Cremers, M. Horvat, J. Hoyland, S. Scott, T. van der Merwe, A comprehensive symbolic analysis of TLS 1.3. In B.M. Thuraisingham, D. Evans, T. Malkin, D. Xu, editors, *ACM CCS 2017*, pp. 1773–1788. ACM Press (2017). https://doi.org/10.1145/3133956.3134063

[30] R. Canetti, S. Halevi, J. Katz, A forward-secure public-key encryption scheme, in E. Biham, editor, *EUROCRYPT 2003, vol. 2656 of LNCS*, pp. 255–271. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_16

[31] C. Cremers, M. Horvat, S. Scott, T. van der Merwe, Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication, in *2016 IEEE Symposium on Security and Privacy*, pp. 470–485. IEEE Computer Society Press (2016). https://doi.org/10.1109/SP.2016.35

[32] S. Chen, S. Jero, M. Jagielski, A. Boldyreva, C. Nita-Rotaru, Secure communication channel establishment: TLS 1.3 (over TCP fast open) vs. QUIC, in K. Sako, S. Schneider, P.Y.A. Ryan, editors, *ESORICS 2019, Part I, vol. 11735 of LNCS*, pp. 404–426. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-29959-0_20

[33] R. Canetti, H. Krawczyk, Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *EUROCRYPT 2001, vol. 2045 of LNCS*, pp. 453–474. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_28

[34] R. Canetti, H. Krawczyk, Security analysis of IKE's signature-based key-exchange protocol, in M. Yung, editor, *CRYPTO 2002, vol. 2442 of LNCS*, pp. 143–161. Springer, Heidelberg (2002). http://eprint.iacr.org/2002/120/. https://doi.org/10.1007/3-540-45708-9_10

[35] Codenomicon. The Heartbleed bug. http://heartbleed.com (2014)

[36] E. Crockett, C. Paquin, D. Stebila, Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH, in *NIST 2nd Post-Quantum Cryptography Standardization Conference 2019* (2019)

[37] T. Dierks, C, Allen, The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard) (1999). Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507, 7919. https://www.rfc-editor.org/rfc/rfc2246.txt, https://doi.org/10.17487/RFC2246

[38] B. Dowling, M. Fischlin, F. Günther, D. Stebila, A cryptographic analysis of the TLS 1.3 handshake protocol candidates, in I. Ray, N. Li, C. Kruegel, editors, *ACM CCS 2015*, pp. 1197–1210. ACM Press (2015). https://doi.org/10.1145/2810103.2813653

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

[40] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Zanella-Béguelin, K. Bhargavan, J. Pan, J.K. Zinzindohoue, Implementing and proving the TLS 1.3 record layer, in *2017 IEEE Symposium on Security and Privacy*, pp. 463–482. IEEE Computer Society Press (2017). https://doi.org/10.1109/SP.2017.58

[41] A. Delignat-Lavaud, C. Fournet, B. Parno, J. Protzenko, T. Ramananandro, J. Bosamiya, J. Lallemand, I. Rakotonirina, Y. Zhou, A security model and fully verified implementation for the IETF QUIC record layer. Cryptology ePrint Archive, Report 2020/114 (2020). https://eprint.iacr.org/2020/114

[42] C.D. de Saint Guilhem, M. Fischlin, B. Warinschi, Authentication in key-exchange: Definitions, relations and composition. Cryptology ePrint Archive, Report 2019/1203 (2019). https://eprint.iacr.org/2019/1203

[43] H. Davis, F. Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols, in *19th International Conference on Applied Cryptography and Network Security, ACNS 2021*, 2021. To appear. Available as Cryptology ePrint Archive, Report 2020/1029. https://eprint.iacr.org/2020/1029

[44] N. Drucker, S. Gueron, Selfie: reflections on TLS 1.3 with PSK. *Journal of Cryptology* (2021). To appear. Available as Cryptology ePrint Archive, Report 2019/347. https://eprint.iacr.org/2019/347.

[45] D. Diemert, T. Jager, On the tight security of TLS 1.3: theoretically-sound cryptographic parameters for real-world deployments. *Journal of Cryptology* (2021). To appear. Available as Cryptology ePrint Archive, Report 2020/726. https://eprint.iacr.org/2020/726

[46] D. Derler, T. Jager, D. Slamanig, C. Striecks, Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In J.B. Nielsen, V. Rijmen, editors, *EUROCRYPT 2018, Part III, vol. 10822 of LNCS*, pp. 425–455. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-78372-7_14

[47] B. Dowling, *Provable Security of Internet Protocols*. Ph.D. thesis, Queensland University of Technology, Brisbane, Australia (2017). http://eprints.qut.edu.au/108960/

[48] T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507, 7919. https://www.rfc-editor.org/rfc/rfc4346.txt. https://doi.org/10.17487/RFC4346

[49] T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919. https://www.rfc-editor.org/rfc/rfc5246.txt, https://doi.org/10.17487/RFC5246

[50] B. Dowling, D. Stebila, Modelling ciphersuite and version negotiation in the TLS protocol, in E. Foo, D. Stebila, editors, *ACISP 15, vol. 9144 of LNCS*, pp. 270–288. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-19962-7_16

[51] T. Duong. BEAST. http://vnhacker.blogspot.com.au/2011/09/beast.html (2011)

[52] M. Fischlin, F. Günther, Multi-stage key exchange and the case of Google's QUIC protocol, in G.-J. Ahn, M. Yung, N. Li, editors, *ACM CCS 2014*, pp. 1193–1204. ACM Press (2014). https://doi.org/10.1145/2660267.2660308

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

[54] M. Fischlin, F. Günther, G.A. Marson, K.G. Paterson, Data is a stream: Security of stream-based channels, in R. Gennaro, M.J.B. Robshaw, editors, *CRYPTO 2015, Part II, vol. 9216 of LNCS*, pp. 545–564. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_27

[55] M. Fischlin, F. Günther, B. Schmidt, B. Warinschi, Key confirmation in key exchange: a formal treatment and implications for TLS 1.3, in *2016 IEEE Symposium on Security and Privacy*, pp. 452–469. IEEE Computer Society Press (2016). https://doi.org/10.1109/SP.2016.34

[56] S. Gajek, A universally composable framework for the analysis of browser-based security protocols, in J. Baek, F. Bao, K. Chen, X. Lai, editors, *ProvSec 2008, vol. 5324 of LNCS*, pp. 283–297. Springer, Heidelberg (2008)

[57] F. Günther, B. Hale, T. Jager, S. Lauer, 0-RTT key exchange with full forward secrecy, in J.-S. Coron, J.B. Nielsen, editors, *EUROCRYPT 2017, Part III, vol. 10212 of LNCS*, pp. 519–548. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-56617-7_18

[58] F. Giesen, F. Kohlar, D. Stebila, On the security of TLS renegotiation, in A.-R. Sadeghi, V.D. Gligor, M. Yung, editors, *ACM CCS 2013*, pp. 387–398. ACM Press (2013). https://doi.org/10.1145/2508859.2516694

[59] M.D. Green, I. Miers, Forward secure asynchronous messaging from puncturable encryption, in *2015 IEEE Symposium on Security and Privacy*, pp. 305–320. IEEE Computer Society Press (2015). https://doi.org/10.1109/SP.2015.26

[60] F. Günther, S. Mazaheri, A formal treatment of multi-key channels, in J. Katz, H. Shacham, editors, *CRYPTO 2017, Part III, vol. 10403 of LNCS*, pp. 587–618. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-63697-9_20

[61] F. Günther, *Modeling advanced security aspects of key exchange and secure channel protocols*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany (2018). http://tuprints.ulb.tu-darmstadt.de/7162/

[62] B. Hale, T. Jager, S. Lauer, J. Schwenk, Simple security definitions for and constructions of 0-RTT key exchange, in D. Gollmann, A. Miyaji, H. Kikuchi, editors, *ACNS 17, vol. 10355 of LNCS*, pp. 20–38. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-61204-1_2

[63] J. Jonsson, B.S. Kaliski Jr., On the security of RSA encryption in TLS, in Moti Yung, editor, *CRYPTO 2002, vol. 2442 of LNCS*, pp. 127–142. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_9

[64] T. Jager, F. Kohlar, S. Schäge, J. Schwenk, On the security of TLS-DHE in the standard model, in R. Safavi-Naini, R. Canetti, editors, *CRYPTO 2012, vol. 7417 of LNCS*, pp. 273–293. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_17

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

[66] T. Jager, J. Schwenk, J. Somorovsky, On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption, in I. Ray, N. Li, C. Kruegel, editors, *ACM CCS 2015*, pp. 1185–1196. ACM Press (2015). https://doi.org/10.1145/2810103.2813657

[67] H. Krawczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational) (1997). Updated by RFC 6151. https://www.rfc-editor.org/rfc/rfc2104.txt, https://doi.org/10.17487/RFC2104

[68] H. Krawczyk, P. Eronen, HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational) (2010). https://www.rfc-editor.org/rfc/rfc5869.txt, https://doi.org/10.17487/RFC5869

[69] M. Kohlweiss, U. Maurer, C. Onete, B. Tackmann, D. Venturi, (De-)constructing TLS 1.3, in A. Biryukov, V. Goyal, editors, *INDOCRYPT 2015, vol. 9462 of LNCS*, pp. 85–102. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-26617-6_5

[70] H. Krawczyk, K.G. Paterson, H. Wee, On the security of the TLS protocol: a systematic analysis. In R. Canetti, J.A. Garay, editors, *CRYPTO 2013, Part I, vol. 8042 of LNCS*, pp. 429–448. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_24

[71] H. Krawczyk, The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *CRYPTO 2001, vol. 2139 of LNCS*, pp. 310–331. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_19

[72] H. Krawczyk, SIGMA: the "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols, in D. Boneh, editor, *CRYPTO 2003, vol. 2729 of LNCS*, pp. 400–425. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_24

[73] H. Krawczyk, Cryptographic extraction and key derivation: the HKDF scheme, in T. Rabin, editor, *CRYPTO 2010, vol. 6223 of LNCS*, pp. 631–648. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_34

[74] H. Krawczyk, [IETF TLS mailing list] Re: Call for consensus: Removing DHE-based 0-RTT. https://mailarchive.ietf.org/arch/msg/tls/xmnvrKEQkEbD-u8HTeQkyitmclY (2016)

[75] H. Krawczyk, A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in TLS 1.3). In E.R. Weippl, S. Katzenbeisser, C. Kruegel, A.C. Myers, S. Halevi, editors, *ACM CCS 2016*, pp. 1438–1450. ACM Press (2016). https://doi.org/10.1145/2976749.2978325

[76] F. Kohlar, S. Schäge, and J. Schwenk, On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367 (2013). http://eprint.iacr.org/2013/367

[77] H. Krawczyk, H. Wee, The OPTLS protocol and TLS 1.3, in *2016 IEEE European Symposium on Security and Privacy*, pp. 81–96. IEEE (2016). https://doi.org/10.1109/EuroSP.2016.18

[78] R. Lychev, S. Jero, A. Boldyreva, C. Nita-Rotaru, How secure and quick is QUIC? Provable security and performance analyses, in *2015 IEEE Symposium on Security and Privacy*, pp. 214–231. IEEE Computer Society Press (2015). https://doi.org/10.1109/SP.2015.21

[79] B.A. LaMacchia, K. Lauter, A. Mityagin, Stronger security of authenticated key exchange, in W. Susilo, J.K. Liu, Y. Mu, editors, *ProvSec 2007, vol. 4784 of LNCS*, pp. 1–16. Springer, Heidelberg (2007)

[80] A. Luykx, K.G. Paterson. Limits on authenticated encryption use in TLS (2017). http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf

[81] Y. Li, S. Schäge, Z. Yang, F. Kohlar, J. Schwenk, On the security of the pre-shared key ciphersuites of TLS, in H. Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pp. 669–684. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_38

[82] X. Li, J. Xu, Z. Zhang, D. Feng, H. Hu, Multiple handshakes security of TLS 1.3 candidates, in *2016 IEEE Symposium on Security and Privacy*, pp. 486–505. IEEE Computer Society Press (2016). https://doi.org/10.1109/SP.2016.36

[83] C. MacCárthaigh, [IETF TLS mailing list] Security review of TLS1.3 0-RTT. https://mailarchive.ietf.org/arch/msg/tls/mHxi-O3du9OQHkc6CBWBpc_KBpA (2017)

[84] B. Möller, T. Duong, K. Kotowicz, This POODLE bites: exploiting the SSL 3.0 fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf (2014)

[85] G.A. Marson, B. Poettering, Security notions for bidirectional channels. *IACR Trans. Symm. Cryptol.*, **2017**(1):405–426 (2017). https://doi.org/10.13154/tosc.v2017.i1.405-426

[86] P. Morrissey, N.P. Smart, B. Warinschi, A modular security analysis of the TLS handshake protocol, in J. Pieprzyk, editor, *ASIACRYPT 2008, vol. 5350 of LNCS*, pp. 55–73. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_5

[87] K.G. Paterson, T. Ristenpart, T. Shrimpton, Tag size does matter: attacks and proofs for the TLS record protocol, in D.H. Lee, X. Wang, editors, *ASIACRYPT 2011, vol. 7073 of LNCS*, pp. 372–389. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_20

[88] C. Patton, T. Shrimpton, Partially specified channels: the TLS 1.3 record layer without elision, in D. Lie, M. Mannan, M. Backes, X. Wang, editors, *ACM CCS 2018*, pp. 1415–1428. ACM Press (2018). https://doi.org/10.1145/3243734.3243789

[89] K.G. Paterson, T. van der Merwe, Reactive and proactive standardisation of TLS, in L. Chen, D.A. McGrew, C.J. Mitchell, editors, *Security Standardisation Research: Third International Conference (SSR 2016)*, volume 10074 of *Lecture Notes in Computer Science*, pp. 160–186, Gaithersburg, MD, USA, December 5–6 (2016). Springer

[90] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard) (2018). https://www.rfc-editor.org/rfc/rfc8446.txt, https://doi.org/10.17487/RFC8446

[91] P. Rogaway, Formalizing human ignorance, in P.Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06, vol. 4341 of LNCS*, pp. 211–228. Springer, Heidelberg (2006)

[92] E. Rescorla, H. Tschofenig, N. Modadugu, The Datagram Transport Layer Security (DTLS) Protocol Version 1.3 – draft-ietf-tls-dtls13-33. https://tools.ietf.org/html/draft-ietf-tls-dtls13-33 (2019)

[93] P. Schwabe, D. Stebila, T. Wiggers, Post-quantum TLS without handshake signatures, in J. Ligatti, X. Ou, J. Katz, G. Vigna, editors, *ACM CCS 20*, pp. 1461–1480. ACM Press (2020). https://doi.org/10.1145/3372297.3423350

[94] M. Thomson, S. Turner, Using TLS to Secure QUIC – draft-ietf-quic-tls-29. https://tools.ietf.org/html/draft-ietf-quic-tls-29 (2020)