A Cryptographic Analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol

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

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

January 29, 2016

Abstract. We analyze the handshake protocol of TLS 1.3 `draft-ietf-tls-tls13-10` (published October 2015). This continues and extends our previous analysis (CCS 2015, Cryptology ePrint Archive 2015) of former TLS 1.3 drafts (`draft-ietf-tls-tls13-05` and `draft-ietf-tls-tls13-dh-based`). Here we show that the full (EC)DHE Diffie–Hellman-based handshake of `draft-10` is also secure in the multi-stage key exchange framework of Fischlin and Günther which captures classical Bellare–Rogaway key secrecy for key exchange protocols that derive multiple keys.

We also note that a recent protocol change—the introduction of a `NewSessionTicket` message for resumption, encrypted under the application traffic key—impairs the protocol modularity and hence our compositional guarantees that ideally would allow an independent analysis of the record protocol. We additionally analyze the pre-shared key modes (with and without ephemeral Diffie–Hellman key), and fit them into the composability framework, addressing composability with the input resumption secret from a previous handshake and of the output session keys.

1 Introduction

The Transport Layer Security (TLS) working group of the Internet Engineering Task Force (IETF) is currently on its way to standardizing the next TLS 1.3 version, as a result of years of discussion, improvement, and in response to detected weaknesses and design problems in previous TLS versions.

1.1 Our Prior Results

Earlier in 2015 [DFGS15a], we cryptographically analyzed two intermediate drafts of TLS 1.3, `draft-ietf-tls-tls13-05` (which we shorten to `draft-05`, [Res15a]) and `draft-ietf-tls-tls13-dh-based` (short: `draft-dh`, [Res15c]). (As this paper is targeted to the expert TLS audience of the TRON ("TLSv1.3 - Ready or Not?") workshop[1], we skip repeating a detailed introduction and "dive right in". The full version of our earlier work is available on the IACR Cryptology ePrint Archive [DFGS15b].) In that work, we extended the game-based multi-stage key exchange model by Fischlin and Günther [FG14] (itself following the paradigm of the Bellare–Rogaway model [BR94]), where a key exchange derives not only one but multiple keys, to handle unauthenticated sessions, different authentication modes in parallel, and key exchanges from pre-shared symmetric keys. We showed that the primary full Diffie–Hellman-based handshake as well

[1] `https://www.internetsociety.org/events/ndss-symposium-2016/tron-workshop-call-papers`

as the resumption handshake achieved key secrecy in the multi-stage setting: for TLS 1.3, this means key secrecy of the handshake traffic key, the application traffic key, as well as the resumption and exporter master secrets. We note that our prior work did not analyze the 0-RTT handshake mode of TLS 1.3, since it was not fully specified at the time of writing of our earlier work.

As a second component of our preliminary work, we furthermore augmented the composition frameworks by Brzuska et al. [BFWW11] and Fischlin and Günther [FG14] to encompass protocols (like TLS 1.3) in which unauthenticated, unilaterally authenticated, and mutually authenticated sessions run concurrently. This generic composition result enables the independent analysis of the record protocol security, ensuring that, in particular, the final keys established in the full handshake (i.e., the application traffic key, the resumption master secret, and the exporter master secret) can safely be used in any symmetric-key protocol.

We refer to [DFGS15b] for the full details of the multi-stage key exchange model and the composition result which we again use for our analysis of `draft-10`.

1.2 This Paper's Results

This work combines the contributions from our earlier analysis of TLS 1.3 handshake candidates `draft-05` and `draft-dh` with an updated analysis that captures the modifications in the TLS 1.3 draft `draft-ietf-tls-tls13-10` [Res15b] (which we shorten to `draft-10`), published in October 2015.

On a high level, we are able to confirm that the (multi-stage) key secrecy guarantees provided by the earlier analyzed handshake candidates carry over to `draft-10`, mainly because `draft-10` adopts the `draft-dh` handshake variant (which we analyzed) and refines its key schedule (largely following the OPTLS protocol design invented and analyzed by Krawczyk and Wee [KW15, KW16]), while maintaining the main handshake structure and strong key separation of the previous drafts. More specifically, we ascertain that the full (EC)DHE handshake still satisfies the notion of multi-stage key secrecy, providing in particular forward secrecy for all of the derived keys. Unfortunately, the modular analysis is compromised—in particular composition for the application transport key is not possible in its generic form due to the introduction of a new message called `NewSessionTicket` in `draft-10`.

Furthermore, we analyze the pre-shared key handshake modes (PSK and PSK-DHE) of `draft-10`, which replaced the former resumption handshake in `draft-05`. Both protocols start with the previously established keys to derive the new keys, but the PSK-DHE version adds another Diffie-Hellman step on top. We are able to show that the PSK handshake still achieves key secrecy guarantees without forward secrecy as in the previously analyzed `draft-05`. For the PSK-DHE handshake, we confirm that the combination of pre-shared and ephemeral Diffie–Hellman keys indeed additionally achieves the desired forward secrecy. This in particular also renders the application traffic key established in the PSK-DHE handshake amenable to our composition result, allowing an independent security analysis of its use in the record protocol.

We next discuss our results in more detail.

Security of the `draft-10` full (EC)DHE handshake. In this work, we show that the full (EC)DHE handshake of `draft-10` is a secure multi-stage key exchange protocol where different stages and simultaneous runs of the protocols can be unauthenticated, unilaterally authenticated, or mutually authenticated. On a high level, this means that the handshake establishes record layer as well as resumption and exporter keys that look random to an adversary. This holds even for sessions that run concurrently and if the adversary controls the network, is able to corrupt the long-term secret keys of other parties, and allowed to reveal keys established in other sessions, thus providing quite strong security guarantees for practice. Moreover, using the multi-stage model allows us to show that even leakage of record layer or exporter keys in the same handshake session do not compromise each other's security.

Notably, our security proof only relies on so-called standard cryptographic assumptions such as the Decisional Diffie–Hellman (DDH) assumption, unforgeability of the deployed signature scheme, collision resistance of the hash function, and pseudorandomness of the HKDF key derivation function. This is in sharp contrast to many other key exchange protocols, where often the key derivation function is modeled as a random oracle. The cryptographic analysis of signed-Diffie–Hellman ciphersuites in TLS 1.2 required an uncommon (yet not implausible) pseudorandom-oracle Diffie–Hellman assumption [JKSS12, KPW13], but this is not required for `draft-10`.

Security of the `draft-10` PSK and PSK-DHE handshakes. We also analyze the pre-shared key handshake modes of `draft-10`, PSK and PSK-DHE, and show that they as well are secure multi-stage (preshared-secret) key exchange protocols (relying on the unforgeability of the HMAC message authentication code instead of signature unforgeability for authentication). The two pre-shared key modes differ in that the plain PSK handshake does not achieve forward secrecy while the PSK-DHE handshake, mixing fresh ephemeral Diffie–Hellman keys into the key derivation, does indeed establish forward-secret keys as envisioned. For the latter analysis, we extend the multi-stage preshared-secret key exchange model formalized in our previous work [DFGS15a] to capture forward secrecy in the setting of pre-shared keys.

Composition with the record layer and the role of `NewSessionTicket`. When it comes to the overall security of TLS 1.3, we follow a compositional approach. More specifically, we show that any final key (or in other words: any key not used in the handshake itself) established in a multi-stage key exchange that enjoys, primarily, forward secrecy and key independence (a certain technical form of computational key separation), can safely be used in *any* subsequent symmetric-key protocol. Via this generic composition result we can deduce such guarantees for the resumption and exporter master secret of the full handshake as well as the application traffic key and the exporter master secret of the pre-shared key modes of `draft-10`. This means that, in particular, the usage of the resumption master secret of the full handshake as an input to later PSK/PSK-DHE handshake runs is safe. Likewise, it allows an independent analysis of the record layer using the application traffic key established in the forward-secret PSK-DHE mode (which was not possible in the non-forward secret session resumption mode of `draft-05` analyzed in our previous work).

Regrettably, this modular approach of analyzing the handshake protocol and the protocol using an established key independently cannot be applied to the use of the application traffic key derived in the full handshake. The reason for this is that the application traffic key is (potentially) already used in the full handshake to encrypt a `NewSessionTicket` message which is used for session resumption. Though envisioned by the designers as a "post-handshake message," sending `NewSessionTicket` thus implies that the application traffic key is effectively used *within* the handshake protocol, which revives an old obstacle for modeling the handshake's security known from the formal security analyses of the previous TLS version 1.2 (e.g., [MSW08]). There, usage of the derived session key to encrypt the handshake's `Finished` messages rendered classical key secrecy notions (in the style of Bellare and Rogaway [BR94]) unachievable, leading to a monolithic analysis of the handshake and record layer together [JKSS12].

While (multi-stage) key secrecy itself is not affected by the `NewSessionTicket` message in `draft-10`, the generic compositional guarantees for the application traffic key fall prey to this change. (As noted above, composition involving all other final keys—the resumption and exporter master secrets, as well as all output keys in the PSK-DHE handshake—is not affected.) Essentially, the `NewSessionTicket` violates a strict logical separation between the handshake and the record protocol for the full handshake's application traffic key, which is reflected in our model as a loss of modularity by rendering generic compositional guarantees unachievable for that key. Our compositional technique not being applicable anymore prevents a modular, independent security analysis of the record protocol along these compositional lines and might necessitate a more complex, entangled analysis of the combined handshake and record protocol as for previous TLS versions (e.g., [JKSS12]). We therefore highly recommend to reestablish a logical and cryptographic separation between the handshake establishing keys and the usage of keys in the record protocol, and discuss several options to achieve this in Section 3.

Limitations. Naturally, our analysis is limited to the specification in `draft-ietf-tls-tls13-10` [Res15b]: our work hence serves as a cryptographic insight into the draft design, but cannot be a definitive analysis of the final TLS 1.3 protocol.

In this work we raise our results for the full and pre-shared key handshakes to the latest `draft-10` specification, but do not capture the fourth handshake mode which allows for a zero round-trip time (0-RTT) key exchange.

1.3 Related Work

We refer to our earlier paper [DFGS15a] for a detailed history of research on the TLS protocol and only list new work that has appeared since then.

Krawczyk and Wee [KW15, KW16] present the OPTLS protocol that constitutes the clean conceptual foundation of the TLS 1.3 handshake, its full, 0-RTT and pre-shared key modes, as well as its key schedule which enabled our standard-model security results in the first place. Jager et al. [JSS15] describe a cross-ciphersuite-family, cross-protocol-version, and cross-protocol attack on TLS 1.3 and Google's QUIC protocol [Ros13] which leverages Bleichenbacher-style weaknesses in implementations of RSA-based PKCS#1 v1.5 encryption of, e.g., previous TLS versions to forge RSA signatures in TLS 1.3 and QUIC.[2] Bhargavan et al. [BBF$^+$16] discuss downgrade resilience as a formal security notion for key exchange protocols and, in particular, analyze downgrade protection in TLS 1.3 `draft-10` as well as proposed fallback mechanisms in the follow-up version `draft-ietf-tls-tls13-11`.

2 The TLS 1.3 draft-10 Full Handshake Protocol

The TLS 1.3 full handshake protocol is divided into two phases: the *negotiation* phase, where parties negotiate ciphersuites and key-exchange parameters, generate unauthenticated shared key material, and establish handshake traffic keys; and the *authentication* phase, where parties authenticate the handshake transcript according to the authentication properties negotiated earlier and output authenticated application traffic keys, independent from the previous handshake traffic keys.

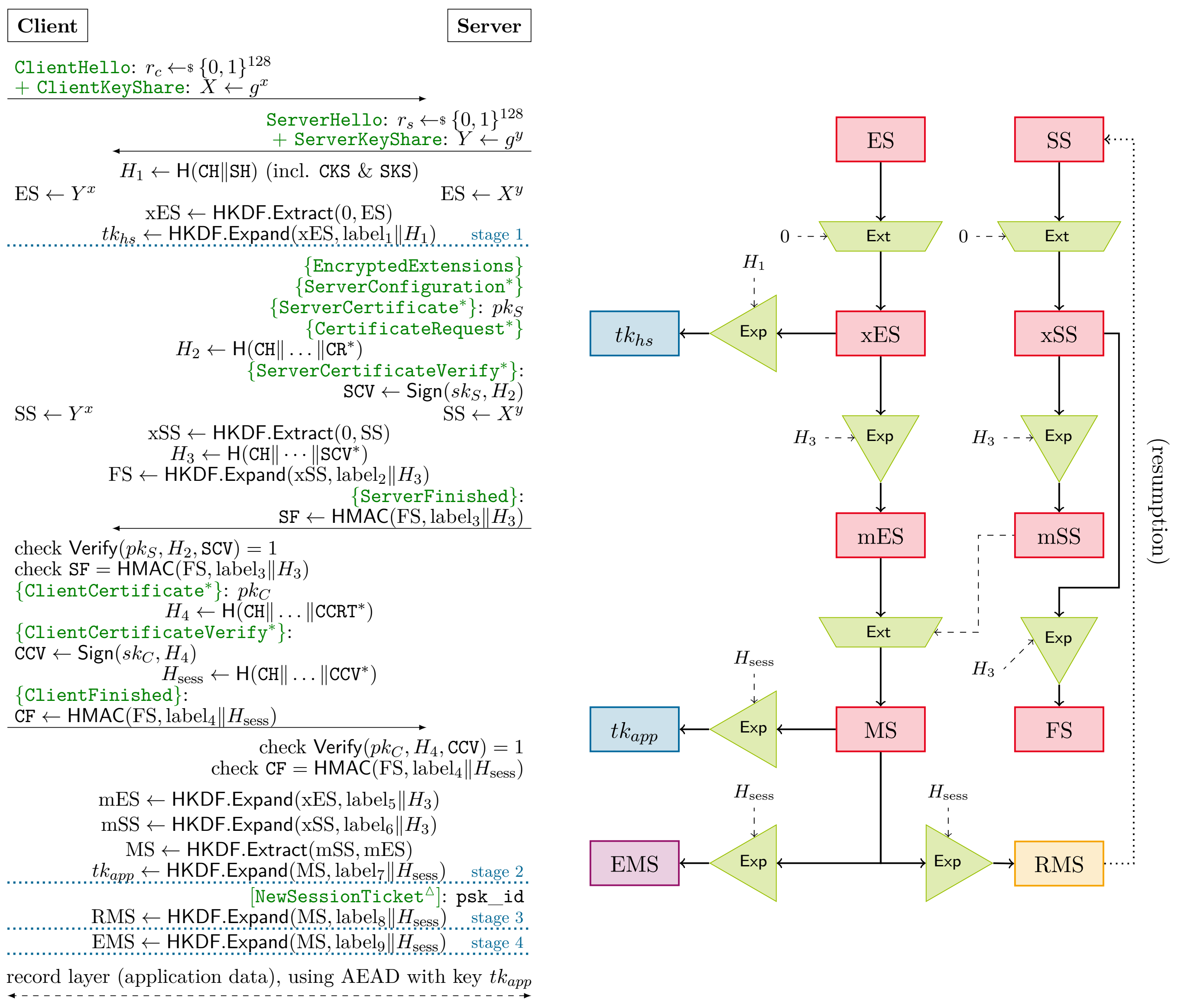
Figure 1 shows the message flow and relevant cryptographic computations as well as the key schedule for the full handshake in `draft-10`. The handshake messages are as follows:

- `ClientHello` (`CH`)/`ServerHello` (`SH`) contain the supported versions and ciphersuites for negotiation purposes, as well as random nonces r_c resp. r_s. Both `CH` and `SH` can also include various extension fields.
- `ClientKeyShare` (`CKS`)/`ServerKeyShare` (`SKS`) are extensions sent within the `ClientHello` resp. `ServerHello` messages which contain the ephemeral Diffie–Hellman shares $X = g^x$ resp. $Y = g^y$ for one or more (in case of the client) groups.

Both parties can now compute the ephemeral secret ES as the Diffie–Hellman shared value g^{xy}. Key derivation is then done using HKDF in the extract-then-expand paradigm [Kra10], computing first an extracted value xES from which the handshake traffic key tk_{hs} is expanded; both are unauthenticated at this point.

We adopt here the standard notation for the two HKDF functions: $\mathsf{HKDF.Extract}(XTS, SKM)$ on input an (non-secret and potentially fixed) extractor salt XTS and some source key material SKM outputs a

[2]The Jager et al. attack does not contradict our provable security analysis since our formalism analyses solely TLS 1.3, and fallback mechanisms are outside of our scope.



Client **Server**

ClientHello: $r_c \leftarrow_{\$} \{0,1\}^{128}$
+ ClientKeyShare: $X \leftarrow g^x$

ServerHello: $r_s \leftarrow_{\$} \{0,1\}^{128}$
+ ServerKeyShare: $Y \leftarrow g^y$

$H_1 \leftarrow \mathsf{H}(\mathtt{CH}\|\mathtt{SH})$ (incl. `CKS` & `SKS`)
$\mathrm{ES} \leftarrow Y^x$ $\mathrm{ES} \leftarrow X^y$
$\mathrm{xES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{ES})$
$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_1\|H_1)$ stage 1

{EncryptedExtensions}
{ServerConfiguration*}
{ServerCertificate*}: pk_S
{CertificateRequest*}
$H_2 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CR}^*)$
{ServerCertificateVerify*}:
$\mathtt{SCV} \leftarrow \mathsf{Sign}(sk_S, H_2)$
$\mathrm{SS} \leftarrow Y^x$ $\mathrm{SS} \leftarrow X^y$
$\mathrm{xSS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{SS})$
$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\|\cdots\|\mathtt{SCV}^*)$
$\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_2\|H_3)$
{ServerFinished}:
$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3\|H_3)$

check $\mathsf{Verify}(pk_S, H_2, \mathtt{SCV}) = 1$
check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3\|H_3)$
{ClientCertificate*}: pk_C
$H_4 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CCRT}^*)$
{ClientCertificateVerify*}:
$\mathtt{CCV} \leftarrow \mathsf{Sign}(sk_C, H_4)$
$H_{\mathrm{sess}} \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{CCV}^*)$
{ClientFinished}:
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4\|H_{\mathrm{sess}})$

check $\mathsf{Verify}(pk_C, H_4, \mathtt{CCV}) = 1$
check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4\|H_{\mathrm{sess}})$

$\mathrm{mES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_5\|H_3)$
$\mathrm{mSS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_6\|H_3)$
$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{mSS}, \mathrm{mES})$
$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_7\|H_{\mathrm{sess}})$ stage 2
[NewSessionTicket$^\triangle$]: `psk_id`
$\mathrm{RMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_8\|H_{\mathrm{sess}})$ stage 3
$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_9\|H_{\mathrm{sess}})$ stage 4

record layer (application data), using AEAD with key tk_{app}

Figure 1: The full (EC)DHE handshake protocol in TLS 1.3 **`draft-10`** (left) and its key schedule (right).
`XXX`: Y denotes TLS message **`XXX`** containing Y. `{XXX}` resp. `[XXX]` indicate a message **`XXX`** encrypted using AEAD encryption with handshake traffic key tk_{hs} resp. application traffic key tk_{app}. + `XXX` indicates a message that is sent as an extension within the previous message. `XXX`* indicates a message that is only sent in unilateral or mutual authentication modes. `XXX`$^\triangle$ indicates a message that is only sent when later resumption shall be allowed.
In the key schedule, Ext and Exp are short for $\mathsf{HKDF.Extract}$ resp. $\mathsf{HKDF.Expand}$. Dotted-line input to Ext is the (extractor) salt, dotted-line input to Exp is the (context) information input; label inputs (which are distinct for each Exp application) are omitted.

pseudorandom key PRK. $\mathsf{HKDF.Expand}(PRK, CTXinfo)$ on input a pseudorandom key PRK (from the $\mathsf{Extract}$ step) and some (potentially empty) context information $CTXinfo$ outputs key material KM.[3]

All subsequent messages are encrypted using tk_{hs}:

- `EncryptedExtensions` (`EE`) contains more extensions.
- `ServerConfiguration` (`SC`) contains a server configuration (cryptographically an additional semi-

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

static Diffie–Hellman share) which allows a client to later run an abbreviated (0-RTT) handshake.

- `ServerCertificate` (`SCRT`)/`ClientCertificate` (`CCRT`) contain the public-key certificate of the respective party.
- `CertificateRequest` (`CR`) indicates the server requests that the client authenticates using a certificate.
- `ServerCertificateVerify` (`SCV`)/`ClientCertificateVerify` (`CCV`) contain a digital signature over the *handshake hash* (the hash of all handshakes messages sent and received at that point in the protocol run).
- `ClientFinished` (`CF`)/`ServerFinished` (`SF`) contain a message authentication code (an HMAC value) computed over the session hash keyed with the finished secret FS. The finished secret in turn is derived from the extracted version xSS of the static secret SS. While the static secret takes different values in other handshake variants, it equals the ephemeral secret (SS = ES) in the full (EC)DHE handshake.

Both parties can now compute the master secret MS (as an extraction of expanded ephemeral and static secrets). From the master secret, the application traffic key tk_{app} as well as the resumption master secrets RMS for use in future session resumptions and the exporter master secret EMS allowing the potential derivation of further keying material are computed through HKDF expansion steps which include the final handshake hash value, which is called the *session hash* H_{sess}.

Finally, an additional message can optionally be sent encrypted using tk_{app}:

- `NewSessionTicket` (`NST`) contains an identifier (a "ticket") for the derived resumption master secret that the client can use the purpose of later session resumption.

3 Comments on the TLS 1.3 draft-10

Several of the comments from our previous work on the design choices in `draft-dh` apply to `draft-10` as well. `draft-10` continues to achieve its main cryptographic goals, including (session-)key independence and privacy of (the key used for) encrypted handshake messages. We previously noted that proofs were made easier by the choice to sign the session hash (the hash of the full transcript), and this continues to apply.

Our main new comment regarding `draft-10` focuses on the new `NewSessionTicket` message and how it affects key separation and, hence, composability.

3.1 Issues with the `NewSessionTicket` Message

As seen in Figure 1, the full `draft-10` handshake includes a `NewSessionTicket` message that is encrypted under the application traffic key tk_{app} and which the server may optionally send at the end of the handshake to issue a pre-shared key identifier which can be used with the resumption master secret for session resumption in a subsequent pre-shared key handshake.

As mentioned in the introduction, the usage of tk_{app} to encrypt the `NewSessionTicket` (`NST`) message within the handshake is similar to how TLS 1.2 and prior versions use the established session key to encrypt the `Finished` messages which precludes classical Bellare–Rogaway key secrecy. This brings back a conceptual problem similar to the one that analyses of previous TLS versions faced: using the established key already in the handshake negatively affects key secrecy in the sense of Bellare and Rogaway, or at least composability. An adversary who is given either the real session key or a random key can test whether they are consistent with the `Finished` messages (in the case of TLS 1.2) or the encryption of `NST` (in the case of `draft-10`). While at first glance one might attempt to treat this as a special first message in the record protocol, this breaks a strict separation between the handshake and record protocols. Moreover, `NST` is conceptually "part of" the handshake as it establishes the identifier for the resumption master secret.

Our multi-stage key exchange model captures this problem, albeit in a slightly different style. In our model, when a session's key is established, the adversary is prompted to decide whether this session should be tested or not. If it is to be a test session, the session key is set to be either real or random, either choice made with equal probability. This value is given to the adversary, and then the protocol continues, using this specific value through the rest of the protocol. In this sense, the *subsequent* use of the session key (as in the case of the `NewSessionTicket` message here) will remain consistent with the value the adversary receives in our multi-stage scenario. Hence, we do not immediately run into the problem faced above. However, if the test value is actually used within the protocol, the corresponding session key becomes no longer *composable*, preventing an independent analysis of its usage in a subsequent symmetric-key protocol, e.g., the encryption of application data.

Admittedly, as for the key usage in the finished message of TLS 1.2, there is no immediate attack vector arising from this approach. It rather constitutes a violation of the modularity of handshake and record protocols in the protocol design (which are supposed to be linked solely via the keys output by the handshake). This violation consequently translates to a break of modularity (i.e., composition) in our model. While it is possible to achieve multi-stage key secrecy (BR-style) by considering `NewSessionTicket` as message in the third stage establishing RMS (i.e., being sent after stage-2 key tk_{app} was established), there is no hope to achieve generically secure composition due to the composed protocol (the record layer employing tk_{app}) now already being used within the handshake. This in particular impedes a clean, independent analysis of the record protocol, as such a result cannot be immediately combined with our full handshake security result for the application traffic key tk_{app}. Instead, the `draft-10` design for this case would have to be analyzed using a monolithic approach such as ACCE [JKSS12].

3.2 Alternatives for the `NewSessionTicket` Message

We consider several options to salvage the compositional guarantees for the application traffic key tk_{app} (and preserving those of the resumption and exporter master secret RMS and EMS). We refrain from advocating a particular option, as balancing the engineering constraints may be best left to the TLS working group.

1. **Send `NewSessionTicket` earlier in the handshake (i.e., within the server's first flight), encrypted under tk_{hs}.**
   This approach precludes certain usage scenarios for the `NewSessionTicket` message. In particular, the message cannot depend on the final (server's) session state anymore, which rules out tickets that encode this state or the resumption master secret RMS as a self-encrypted and self-authenticated value [Res15b, Section 6.3.11], [SZET08, Section 4].
2. **Send `NewSessionTicket` as the final message, but encrypt it under tk_{hs}.**
   Not being contained in the session hashes signed by the server, nor confirmed in its `Finished` message, the `NewSessionTicket` message in this case would not be explicitly authenticated (as tk_{hs} is an unauthenticated key). However, as the `CertificateVerify` signatures comprise tk_{hs} and serve as a retrospective authentication, `NewSessionTicket` can be considered implicitly authenticated at the end of the handshake. Moreover, `NewSessionTicket` is only a pointer to the established resumption master secret RMS, which itself carries the full authentication of the handshake.
3. **Send `NewSessionTicket` as the final message, but encrypt it under a separately derived key tk_{nst}.**
   To achieve the same authentication properties as with sending the `NewSessionTicket` message encrypted under tk_{app}, the cryptographically cleanest approach would be to derive an independent traffic key tk_{nst} for that purpose as $tk_{nst} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}' \| H_{\mathrm{sess}})$, similar to the derivation of tk_{app}, RMS, and EMS, but using a unique label $\mathrm{label}' =$ `"NewSessionTicket key expansion"`.
   Note that, while `NewSessionTicket` is now encrypted under the different, intermediate key tk_{nst}, it does not constitute an additional flight of the server for which the client would have to wait. Indeed, after sending its `Finished` message, the client can immediately switch to tk_{app} for sending data, i.e., activate the client_write_key and client_write_IV components of tk_{app} (cf. [Res15b, Section 7.2]).

For receiving data, the client first switches to (server_write_key and server_write_IV of) tk_{nst}[4] in order to process a potential NewSessionTicket message. After processing this message (or if no NewSessionTicket is sent), the client switches to tk_{app} also for receiving data. We remark that this kind of asynchronous activation of write and read keys is not a new concept, but is already in use in previous TLS versions for the (unilateral) key switches that follow a ChangeCipherSpec message.

We note that the follow-up TLS 1.3 draft-ietf-tls-tls13-11 specifies additional "post-handshake messages", for example for post-handshake (client) authentication. Our third comment above could be extended to envision a separate, cryptographically independent "control channel" for sending these and potentially other post-handshake, non-application data messages.

4 Security of the TLS 1.3 draft-10 Full Handshake

Security of the TLS 1.3 draft-10 full (EC)DHE handshake [Res15b] follows closely along the same lines of argument as for the analysis of the draft-dh handshake candidate in our earlier work [DFGS15a, Section 5]. The underlying security model is basically a multi-stage extension [FG14] of the classical Bellare-Rogaway model [BR94]. It captures the classical key secrecy idea of Bellare and Rogaway in the notion of Multi-Stage security, essentially requiring that derived session keys must be indistinguishable from random strings for an adversary as long as it did not reveal them via, e.g., corrupting one of the parties deriving the key. A second, technical notion, denoted Match security, ensures that the way partnering (two parties' sessions engaging in "the same" key exchange run) is defined via session identifiers for a protocol is sound. The extension to multiple stages includes the distinction between key-dependent and key-independent protocols where the latter refers to protocols in which revealing session keys of some stage does not affect the security of subsequent keys. Another change is the introduction of contributive identifiers which capture executions in which a party has provided all its contribution to the shared key but may not have yet accepted itself, such as the server in the TLS 1.3 full (EC)DHE handshake waiting for the client's final confirmation. Other significant differences are that the protocol may be run in different authentication modes for the various stages, and that forward security may now hold from a certain stage on. We refer to the previous analysis [DFGS15a, DFGS15b] for the comprehensive formal definition of the multi-stage key exchange model.

The necessary changes in the proof for Match security when going from draft-ietf-tls-tls13-05 and draft-ietf-tls-tls13-dh-based to draft-10, as well as in the first two of three proof cases for Multi-Stage security, indeed only reflect some minor changes to the messages being exchanged (in particular, signaling of server configurations for 0-RTT handshakes and the server's signature over the transcript are now split into the two messages ServerConfiguration and ServerCertificateVerify). For the third, main proof case of Multi-Stage security, our adapted version involves two still minor, but more notable changes: First, the full (EC)DHE handshake in draft-10 does not include a semi-static Diffie–Hellman share in the static secret SS anymore, obviating the need for an EUF-CMA signature forgery reduction in this proof case. Second, the extra intermediate expanded ephemeral and static secrets mES and mSS derived (from xES resp. xSS) introduce another reduction step to the PRF security of HKDF.Extract. We provide the technical specification of our model in Appendix A.1 and the adapted, full security analysis in Appendix A.2 for Match security and A.3 for Multi-Stage security.

Theorem 4.1 (Match security of draft-10-full)**.** *The* draft-10 *full handshake is* Match*-secure: for any efficient adversary $\mathcal{A}$ we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-10-full},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|},$$

[4]Note that client_write_key and client_write_IV of tk_{nst} are never used.

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

Theorem 4.2 (Multi-Stage security of `draft-10-full`)**.** *The* `draft-10` *full handshake is* Multi-Stage*-secure in a key-independent and stage-1-forward-secret manner with concurrent authentication properties* $\mathsf{AUTH} = \{(\mathsf{unauth}, \mathsf{unauth}, \mathsf{unauth}, \mathsf{unauth}), (\mathsf{unauth}, \mathsf{unilateral}, \mathsf{unilateral}, \mathsf{unilateral}), (\mathsf{unauth}, \mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual})\}$ *(i.e., no authentication, stage-2 unilateral authentication, and stage-2 mutual authentication). Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_{10}$ such that*

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

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

5 Security of the TLS 1.3 draft-10 Pre-shared Key Handshakes

The TLS 1.3 pre-shared key (PSK) handshake modes are a relatively new addition, having merged session resumption functionalities with earlier pre-shared key handshake variants. There currently exist two PSK handshake variants: one solely based on pre-shared keys (PSK) and one that combines pre-shared keys with an (EC)DHE key exchange (PSK-(EC)DHE); both are shown in Figure 2. Like in the `draft-10` full handshake, the PSK handshake modes are divided into a negotiation and authentication phase. The negotiation phase now offers negotiation of a pre-shared key identifier where the client offers a set of pre-shared key identities previously established with the server (either in an out-of-band manner or as the resumption master secret derived in an earlier full handshake). In contrast to previously analyzed `draft-05` session resumption, the PSK-(EC)DHE handshake variant moreover offers the ability for a client and server sharing a pre-shared key to also negotiate forward-secret keys by including ephemeral (EC)DHE shares as in the full handshake. Key derivation is done as in the full handshake (cf. Figure 1), where, for PSK-(EC)DHE, the static secret SS is the pre-shared secret and the ephemeral secret ES is computed via the unauthenticated key shares. The PSK handshake does not have `ClientKeyShare`/`ServerKeyShare` messages, so it sets both ES and SS to the pre-shared secret. Unlike the `draft-10` full handshake, authentication is not done through signatures. Instead, both parties implicitly authenticate each other via the key derivation over the pre-shared secret, using the MAC tag contained in the `ClientFinished`/`ServerFinished` messages (similarly to `draft-05` session resumption).

We analyze the security of the PSK and PSK-(EC)DHE handshake modes using the Multi-Stage Preshared-Secret Key Exchange model established in our earlier analysis [DFGS15a], which we reproduce in Appendix B. Minor differences exist between the MS-PSKE model introduced in the previous work. The new model includes forward secrecy notions and also differs slightly in how the challenger maintains the list of pre-shared key identifiers and pre-shared secrets. The security of the `draft-10` PSK and PSK-(EC)DHE handshake modes share structural similarities with the security analysis of the `draft-10` full handshake above, as well as the original analysis of the `draft-05` session resumption. There are changes in the proof of Match security to account for the (EC)DHE shares being included in the session identifier, as well as the `ClientPreSharedKey` and `ServerPreSharedKey` messages. The Multi-Stage security proof is also modified to account for the (EC)DHE shares as well as a different key schedule. We provide the technical specification of our model and the full security analysis in Appendix C.

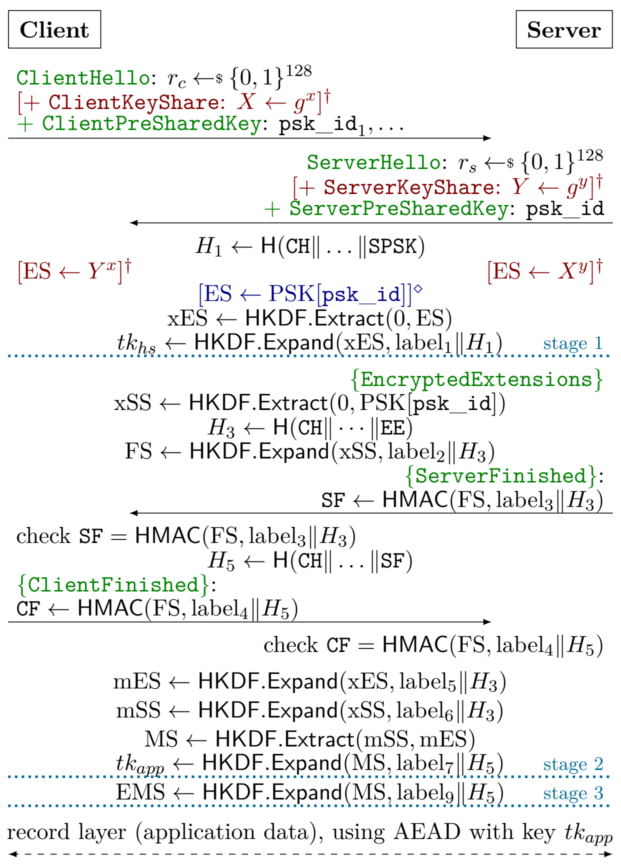

Client
Server
ClientHello: $r_c \leftarrow_\$ \{0,1\}^{128}$
$[+$ ClientKeyShare: $X \leftarrow g^x]^\dagger$
$+$ ClientPreSharedKey: psk_id$_1, \ldots$
ServerHello: $r_s \leftarrow_\$ \{0,1\}^{128}$
$[+$ ServerKeyShare: $Y \leftarrow g^y]^\dagger$
$+$ ServerPreSharedKey: psk_id
$H_1 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SPSK})$
$[\mathrm{ES} \leftarrow Y^x]^\dagger$
$[\mathrm{ES} \leftarrow X^y]^\dagger$
$[\mathrm{ES} \leftarrow \mathrm{PSK}[\mathtt{psk\_id}]]^\diamond$
$\mathrm{xES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{ES})$
$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_1\|H_1)$
stage 1
{EncryptedExtensions}
$\mathrm{xSS} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{PSK}[\mathtt{psk\_id}])$
$H_3 \leftarrow \mathsf{H}(\mathtt{CH}\|\cdots\|\mathtt{EE})$
$\mathrm{FS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_2\|H_3)$
{ServerFinished}:
$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3\|H_3)$
check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_3\|H_3)$
$H_5 \leftarrow \mathsf{H}(\mathtt{CH}\|\ldots\|\mathtt{SF})$
{ClientFinished}:
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4\|H_5)$
check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS}, \mathrm{label}_4\|H_5)$
$\mathrm{mES} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xES}, \mathrm{label}_5\|H_3)$
$\mathrm{mSS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{xSS}, \mathrm{label}_6\|H_3)$
$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{mSS}, \mathrm{mES})$
$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_7\|H_5)$
stage 2
$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_9\|H_5)$
stage 3
record layer (application data), using AEAD with key tk_{app}

Figure 2: The PSK and PSK-(EC)DHE handshake protocol in TLS 1.3 `draft-ietf-tls-tls13-10`. See caption of Figure 1 for notation. Messages/computations only in PSK-(EC)DHE are marked with $[\ldots]^\dagger$. Messages/computations only in PSK (without (EC)DHE) are marked with $[\ldots]^\diamond$. The key schedule is identical to that of Figure 1, except that no RMS is derived.

5.1 Security of PSK-(EC)DHE

The changes between our previous analysis of session resumption and `draft-10-PSK(EC)DHE` in Match security are small, limited to modifications necessary to reflect the additional (EC)DHE shares included in the handshake. For Multi-Stage security, the proof however is markedly different, primarily to deal with the addition of forward secrecy. The previous analysis did not have to be concerned with Corrupt, as the tested session could not be targeted with such queries, and neither could any session sharing the same pss value. The first case must now contend with the scenario where multiple sessions share pre-shared secrets which can be compromised post-acceptance and still expect key secrecy and authentication properties. This introduces the need for extra care in the security analysis in order to replace the affected pre-shared secret across multiple protocol participants in a consistent fashion, leading to an accordingly increased number of proof steps. The other major changes in an additional reduction step to HKDF's security as a pseudorandom function in line with the changes to the key schedule. We provide the theorems and probability statements below, and the adapted full proof of security in Appendix C.2 for Match security and Appendix C.3 for Multi-Stage security.

Theorem 5.1 (Match security of `draft-10-PSK(EC)DHE`). *The* `draft-10-PSK(EC)DHE` *handshake is* Match*-secure: for any efficient adversary $\mathcal{A}$ we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathtt{draft\text{-}10\text{-}PSK(EC)DHE},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|},$$

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

Theorem 5.2 (Multi-Stage security of `draft-10-PSK(EC)DHE`)**.** *The* `draft-10-PSK(EC)DHE` *handshake is* Multi-Stage*-secure in a key-independent and stage-1-forward-secret manner with stage-2 mutual authentication. Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_{10}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \leq 3n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_s^2 \cdot \big(\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_4}\big) \\ & + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5} + n_s \cdot \big(\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{HKDF.Extract},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} \\ & + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_9} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}}\big)\Big),\end{aligned}$$

where n_s is the maximum number of sessions.

5.2 Security of PSK

The security of the PSK handshake follows closely from the security analysis of session resumption in our previous work. The only noticeable change is an additional PRF step in the key schedule. Match security follows nearly verbatim as for `draft-10-PSK(EC)DHE`. For Multi-Stage security, we reproduce the full proof in Appendix C.4.

Theorem 5.3 (Match security of `draft-10-PSK`)**.** *The* `draft-10-PSK` *handshake is* Match*-secure: for any efficient adversary $\mathcal{A}$ we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-10-PSK},\mathcal{A}} \leq n_s^2 \cdot 2^{-|nonce|},$$

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

Theorem 5.4 (Multi-Stage security of `draft-10-PSK`)**.** *The* `draft-10-PSK` *handshake is* Multi-Stage*-secure in a key-independent and non-forward-secret manner with stage-1 mutual authentication. Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_5$ such that*

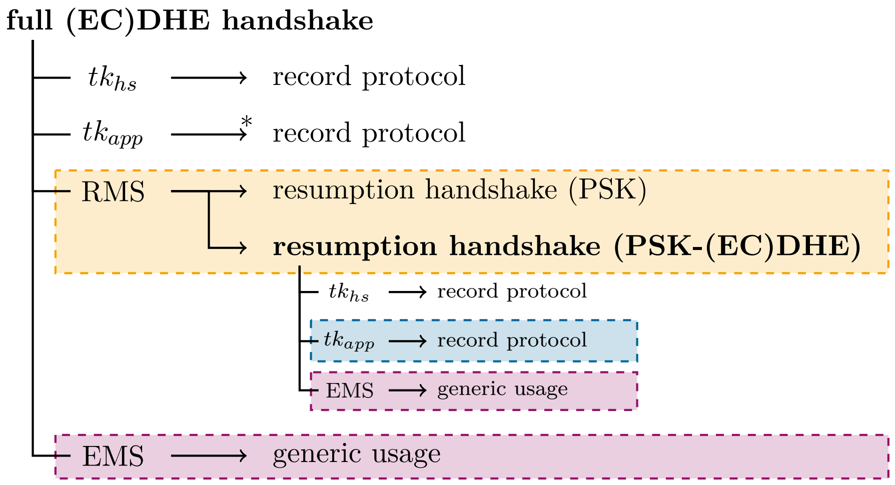
$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-PSK},\mathcal{A}} \leq 3n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_s \cdot \big(\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3} \\ & + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}\big)\Big),\end{aligned}$$

where n_s is the maximum number of sessions.

6 Composition for the Full and PSK Handshakes

In our earlier TLS 1.3 analysis [DFGS15a] we were unable to provide compositional guarantees for the resumption handshake due to its lack of forward secrecy. With our completed multi-stage preshared-secret key exchange model (cf. Appendix B) we can now confirm that our generic composition result extends to the pre-shared key setting for established keys that enjoy forward secrecy, in particular covering the application traffic key and exporter master secret derived in the `draft-10` PSK-(EC)DHE handshake mode. The corresponding proof carries over without change to this setting. Figure 3 illustrates the compositional guarantees we establish for the keys derived in the full and pre-shared key handshakes of `draft-10`.

The corresponding proof carries over without change to this setting, which is why we merely state the augmented composition theorem.



full (EC)DHE handshake
tk_{hs} → record protocol
tk_{app} →* record protocol
RMS → resumption handshake (PSK)
resumption handshake (PSK-(EC)DHE)
tk_{hs} → record protocol
tk_{app} → record protocol
EMS → generic usage
EMS → generic usage

Figure 3: Illustration of the composition result applications in our analysis of the TLS 1.3 `draft-10` full and pre-shared key handshakes. Derived keys are connected to the handshake by solid lines, their usage in protocols is indicated by an arrow. Dashed boxes indicate an application of the composition result to the usage of a specific derived (final) key in the subsequent symmetric-key record or resumption handshake protocol.
* Note that, due to the introduced `NewSessionTicket` message (cf. Section 3), the application traffic key tk_{app} is used within in the full handshake, rendering it non-final and hence unamenable to our generic composition result.

Theorem 6.1 (Multi-stage composition). *Let* KE *be a key-independent stage-j-forward-secret* $\mathsf{Multi\text{-}Stage}$*-secure multi-stage (classical or preshared-secret) key exchange protocol with concurrent authentication properties* AUTH *and key distribution* $\mathcal{D}$ *that allows for efficient multi-stage session matching. Let* Π *be a secure symmetric-key protocol w.r.t. some game* G_Π *with a key generation algorithm that outputs keys with distribution* $\mathcal{D}$*. Then the composition* $\mathsf{KE}_i;\Pi$ *for final stages* $i \geq j$ *is secure w.r.t. the composed security game* $G_{\mathsf{KE}_i;\Pi}$*. Formally, for any efficient adversary* $\mathcal{A}$ *against* $G_{\mathsf{KE}_i;\Pi}$ *there exist efficient algorithms* $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ *such that*

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

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

7 Conclusion

The current version, `draft-ietf-tls-tls13-10`, in principle shows the same cryptographic strength as the previously analyzed versions, `draft-05 draft-dh`. Remarkably, the analyses only rely on standard cryptographic assumptions, forgoing the random oracle model. Furthermore, our analysis provides compositional guarantees for the full (EC)DHE handshake protocol and the PSK-DHE protocol. This means that any security proof of the record layer of TLS 1.3 can be straightforwardly merged to conclude overall security of the combinations of these steps.

On the downside, the new `NewSessionTicket` message introduces similar complications as the `Finished` message in TLS 1.2. Such a mixture of protocol message (i.e., messages protected through the application key) and handshake message impedes a clean analysis, in particular an independent security analysis of the record protocol using the full handshake's application traffic key through our composition result. We have discussed some alternatives to the current deployment of the `NewSessionTicket` message, mainly from a cryptographic point of view, in Section 3.2. We hope that our results in this regard stimulate further discussions about this issue.

Acknowledgments

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

References

[BBF+16] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss, and Santiago Zanella-Béguelin. Downgrade resilience in key-exchange protocols. Cryptology ePrint Archive, Report 2016/072, 2016. https://eprint.iacr.org/2016/072. 4

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 51–62, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. 2

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. 1, 3, 8

[DFGS15a] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15: 22nd Conference on Computer and Communications Security*, pages 1197–1210, Denver, CO, USA, October 12–16, 2015. ACM Press. 1, 3, 4, 8, 9, 11, 15

[DFGS15b] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. Cryptology ePrint Archive, Report 2015/914, 2015. https://eprint.iacr.org/2015/914. 1, 2, 8, 16

[FG14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14: 21st Conference on Computer and Communications Security*, pages 1193–1204, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. 1, 2, 8

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. 3, 7

[JSS15] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15: 22nd Conference on Computer and Communications Security*, pages 1185–1196, Denver, CO, USA, October 12–16, 2015. ACM Press. 4

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. 3

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. 4

[KW15] Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978, 2015. https://eprint.iacr.org/2015/978. 2, 4

[KW16] Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. In *EuroS&P 16: 1st IEEE European Symposium on Security and Privacy*, 2016. (to appear). 2, 4

[MSW08] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 55–73, Melbourne, Australia, December 7–11, 2008. Springer, Heidelberg, Germany. 3

[Res15a] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-05. https://tools.ietf.org/html/draft-ietf-tls-tls13-05, March 2015. 1

[Res15b] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-10. https://tools.ietf.org/html/draft-ietf-tls-tls13-10, October 2015. 2, 4, 7, 8

[Res15c] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-dh-based. https://github.com/ekr/tls13-spec/blob/ietf92_materials/draft-ietf-tls-tls13-dh-based.txt, March 2015. 1

[Ros13] Jim Roskind. QUIC (Quick UDP Internet Connections): Multiplexed Stream Transport Over UDP. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/, December 2013. 4

[SZET08] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077 (Proposed Standard), January 2008. 7

A Security Analysis of the TLS 1.3 draft-10 Full Handshake

A.1 Technical Specification for `draft-10-full` in the Multi-Stage Key Exchange Model

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

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

Here, starred (*) components are not present in all authentication modes.

We capture the further derived resumption master secret RMS and exporter master secret EMS in stages 3 and 4 and define the session identifier to be $\mathsf{sid}_3 = (\mathsf{sid}_2, \text{“RMS”})$ and $\mathsf{sid}_4 = (\mathsf{sid}_2, \text{“EMS”})$ which are uniquely determined by the second-stage identifier sid_2.

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

Concerning the contributive identifiers, we let the client (resp. server) on sending (resp. receiving) the `ClientHello` and `ClientKeyShare` messages set $\mathsf{cid}_1 = (\mathtt{CH}, \mathtt{CKS})$ and subsequently, on receiving (resp. sending) the `ServerHello` and `ServerKeyShare` messages, extend it to $\mathsf{cid}_1 = (\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS})$. The other contributive identifiers are set to $\mathsf{cid}_i = \mathsf{sid}_i$ for stages $i \in \{2, 3, 4\}$ by each party on sending its respective `Finished` message.

As a technical remark, note that the full (EC)DHE handshake of `draft-10` does not involve semi-static keys (from `ServerConfiguration` messages). We hence do not have to treat temporary keys in the notation of our model and can thus ignore NewTempKey queries in the following analysis.

A.2 Proof of `draft-10-full` Match Security

We need to show the six properties of Match security (cf. [DFGS15a, Definition 4.1]).

1. *Sessions with the same session identifier for some stage hold the same session key.*
   For the first stage this follows as the session identifier contains the parties' Diffie–Hellman contributions g^x and g^y, which uniquely identify the Diffie–Hellman key, as well as all data entering the key derivation step. Hence, equal session identifiers imply that both parties compute the same ephemeral secret and the same session key on the first stage. For the second, third, and fourth stage note that the identifier sid_2 (and hence also sid_3 and sid_4) contains the full sid_1, implying that the parties have also computed the same ephemeral secret. Since the key derivation for the stages 2–4 is only based on this secret value (and the identical static secret SS = ES) and data from sid_2, it follows that the session keys must be equal, too.
2. *Sessions with the same session identifier for some stage agree on the authenticity of the stage.*
   Observe that, for the first stage, the only admissible authenticity by design of TLS 1.3 is $\mathsf{auth}_1 = \mathsf{unauth}$ on which, hence, all sessions will agree. For the other stages, the exchanged messages (except for the finished messages) contained in the session identifier sid_2 (and hence also sid_3 and sid_4) uniquely determines the authenticity property for these stages. More precisely, according to the protocol specification, both sessions will agree on $\mathsf{sid}_2 = (\mathtt{ClientHello}, \mathtt{ClientKeyShare}, \mathtt{ServerHello}, \mathtt{ServerKeyShare}, \mathtt{EncryptedExtensions})$ if and only if both have $\mathsf{auth}_2 = \mathsf{unauth}$. If sid_2 additionally contains `ServerConfiguration`* (optional), `ServerCertificate`, and `ServerCertificateVerify`, they agree on $\mathsf{auth}_2 = \mathsf{unilateral}$. If it moreover contains `CertificateRequest`, `ClientCertificate`, and `ClientCertificateVerify`, the sessions agree on mutual authentication. Moreover, $\mathsf{auth}_2 = \mathsf{auth}_3 = \mathsf{auth}_4$ always holds hence same identifiers also imply agreement on authenticity.
3. *Sessions with the same session identifier for some stage share the same contributive identifier.*
   This holds since, for each stage, the contributive identifier value is final and equals the session identifier once the session identifier is set.
4. *Sessions are partnered with the intended partner.*
   First of all observe that this case only applies to unilaterally or mutually authenticated stages, hence the stages 2–4 only. In TLS 1.3, the client obtains the server's identity within the `ServerCertificate` message and vice versa the server obtains the client's identity (in case of mutual authentication) within the `ClientCertificate` message. Moreover, honest clients and servers will not send a certificate attesting an identity different from their own. Hence, as both messages are contained in the session identifiers of stages 2–4 (in the respective authentication mode), agreement on sid_2 (and hence the same for sid_3, sid_4) implies agreement on the respective partner's identity.

5. *Session identifiers are distinct for different stages.*
   This holds trivially as sid_2 contains strictly more messages than sid_1 and sid_3 as well as sid_4 contain unique labels.
6. *At most two sessions have the same session identifier at any stage.*
   Observe that the group element for the Diffie–Hellman key, as well as a random nonce, of both the initiator and the responder enter the session identifiers. Therefore, in order to have a threefold collision among session identifiers of honest parties, the third session would need to pick the same group element and nonce as one of the other two sessions. The probability that there exists such a collision can hence be bounded from above by $n_s^2 \cdot 1/q \cdot 2^{-|nonce|}$ where n_s is the maximum number of sessions, q is the group order, and $|nonce| = 128$ the nonces' bit-length.

$\square$

A.3 Proof of `draft-10-full` Multi-Stage Security

First of all we consider the case that the adversary $\mathcal{A}$ makes a single Test query only. This reduces its advantage, based on a hybrid argument (cf. [DFGS15b, Appendix A]), by a factor at most $1/4n_s$ as there are four stages in each of the n_s sessions. We from now on can speak about *the* session label tested at stage i, which we know in advance.

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

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

This allows us to split the adversary's advantage along these three cases:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-10-full},\mathcal{A}} \leq 4n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{client without partner}}_{\texttt{draft-10-full},\mathcal{A}} \\ & + \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{server without partner}}_{\texttt{draft-10-full},\mathcal{A}} \\ & + \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{test with partner}}_{\texttt{draft-10-full},\mathcal{A}} \Big).\end{aligned}$$

Case A. Test Client without Partner

We first consider the case that the tested session is a client (initiator) session without honest contributive partner in the first stage. Since in the moment a client session can first be tested (i.e., on acceptance of the first key) cid_1 equals sid_1, we know that label also has no session partner in stage 1 (i.e., there is no other label' with $\mathsf{label.sid}_1 = \mathsf{label}'.\mathsf{sid}_1$). Having an honest partner in the second (or later) stage implies having also one in the first stage (as all messages in sid_1 are also contained in $\mathsf{cid}_2 = \mathsf{sid}_2$, $\mathsf{cid}_3 = \mathsf{sid}_3$, and $\mathsf{cid}_4 = \mathsf{sid}_4$), hence the tested session must actually be without honest partner in all stages. Observe that, by the model conditions and sid_1 being set on the client side at the point where K_1 is accepted, the adversary cannot win in this case if the tested key is unauthenticated, hence we can assume that the key is responder-authenticated (i.e., $\mathsf{label.auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$). This allows us to focus on Test queries in the stages 2–4 according to the authentication properties AUTH provided.

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

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

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-10-full},\mathcal{A}} = \mathsf{Adv}^{\text{1-Multi-Stage,client without partner}}_{\texttt{draft-10-full},\mathcal{A}}.$$

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

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

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-10-full},\mathcal{A}} \leq \mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-full},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Game A.2. In this game, we let the challenger abort if the tested client session receives, within the `ServerCertificateVerify` message, a valid signature under the public key pk_U of some user $U \in \mathcal{U}$ such that the hash value has *not* been signed by any of the honest sessions.
Let $\mathsf{abort}_{\mathsf{Sig}}$ denote the event that the challenger aborts in this case. We bound the probability $\Pr[\mathsf{abort}_{\mathsf{Sig}}]$ of its occurrence by the advantage of an adversary $\mathcal{B}_2$ against the $\mathsf{EUF\text{-}CMA}$ security of the signature scheme Sig, denoted $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}$. In the reduction, $\mathcal{B}_2$ first guesses a user $U \in \mathcal{U}$ which it associates with the challenge public key pk^* in the $\mathsf{EUF\text{-}CMA}$ game, then generates all long-term key pairs for the other users $U' \in \mathcal{U} \setminus \{U\}$ and runs the $\mathsf{Multi\text{-}Stage}$ game $G_{A.1}$ for $\mathcal{A}$, including potentially an abort due to hash collisions. For any signature to generate for user U in honest sessions for a hash value, $\mathcal{B}_2$ calls its signing oracle about the hash value. When $\mathsf{abort}_{\mathsf{Sig}}$ is triggered, $\mathcal{B}_2$ outputs the signature the tested client received together with the hash value as a forgery.[5]

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

$$\mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-full},\mathcal{A}} \leq \mathsf{Adv}^{G_{A.2}}_{\texttt{draft-10-full},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2}.$$

Finally, if Game A.2 does not abort, we are assured that an honest session outputs the signature obtained by the tested client session within the `ServerCertificateVerify` message. The signature is

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

computed over $\mathsf{H}(\mathtt{CH}, \mathtt{CKS}, \mathtt{SH}, \mathtt{SKS}, \mathtt{EE}, \mathtt{SC}^*, \mathtt{SCRT}, \mathtt{CR}^*)$, i.e., in particular contains all messages in sid_1. Hence, the tested client and the (distinct) honest session outputting the signature agree on sid_1, so also on cid_1, and are hence (contributively) partnered in the first stage.

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

$$\mathsf{Adv}^{G_{A.2}}_{\texttt{draft-10-full},\mathcal{A}} \leq 0.$$

Case B. Test Server without Partner

We next consider the case that a server (responder) session is tested without honest contributive partner in stage 1. Again, this also implies that there is no honest partner in any of the other stages and, moreover, that also no other session shares the contributive identifiers for stages 2–4 as they include the full first-stage session identifier and thus also cid_1. By definition, the adversary in this case cannot win if the tested key is not mutually authenticated, hence we can assume it is, i.e., $\mathsf{label.auth}_i = \mathsf{mutual}$.

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

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

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-10-full},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{server without partner}}_{\texttt{draft-10-full},\mathcal{A}}.$$

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

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-10-full},\mathcal{A}} = \mathsf{Adv}^{G_{B.1}}_{\texttt{draft-10-full},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_3}.$$

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

$$\mathsf{Adv}^{G_{B.1}}_{\texttt{draft-10-full},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.2}}_{\texttt{draft-10-full},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_4}.$$

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

$$\mathsf{Adv}^{G_{B.2}}_{\texttt{draft-10-full},\mathcal{A}} \leq 0.$$

Case C. Test with Partner

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

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

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{draft-10-full},\mathcal{A}} = \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{test with partner}}_{\texttt{draft-10-full},\mathcal{A}}.$$

Game C.1. Our first modification is to let the challenger abort the game if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H.
We can bound the probability of the game to be aborted by the advantage $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5}$ of an adversary $\mathcal{B}_5$ against the collision resistance of the hash function H. Here, $\mathcal{B}_5$ simply acts as the challenger in Game C.1 and outputs the two distinct input values to H resulting in the same hash value as a collision. It hence holds that

$$\mathsf{Adv}^{G_{C.0}}_{\texttt{draft-10-full},\mathcal{A}} = \mathsf{Adv}^{G_{C.1}}_{\texttt{draft-10-full},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5}.$$

Game C.2. Next, we guess a session $\mathsf{label}' \neq \mathsf{label}$ (among the at most n_s sessions in the game) and abort the game in case this session is not an honest contributive partner (in stage 1) of the tested session, i.e., we abort if $\mathsf{label}.\mathsf{cid}_1 \neq \mathsf{label}'.\mathsf{cid}_1$. Note that we can assume that $\mathcal{A}$ always issues a Test query, as this cannot decrease the adversary's advantage. The guessing strategy then reduces the adversary's advantage by a factor of at most $1/n_s$.

$$\mathsf{Adv}^{G_{C.1}}_{\texttt{draft-10-full},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_{C.2}}_{\texttt{draft-10-full},\mathcal{A}}.$$

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

Game C.3. In Game C.3, we replace the ephemeral secret ES (and, as identical, also the static secret SS) derived in both the tested and (potentially) its contributively partnered session (label resp. label') by a randomly chosen group element $\widetilde{\mathrm{ES}} = \widetilde{\mathrm{SS}} = g^z$ for $z \leftarrow_{\$} \mathbb{Z}_q$.[6]
If $\mathcal{A}$ is able to distinguish this change, we can turn its distinguishing capabilities into an adversary $\mathcal{B}_6$ against the decisional Diffie–Hellman (DDH) assumption in the group $\mathbb{G}$, where $\mathcal{B}_6$ receives values g^u, g^v and h, either $h = g^{uv}$ or h being random. In simulating the game for $\mathcal{A}$, algorithm $\mathcal{B}_6$ uses the challenge group elements g^u and g^v as a replacement for the values g^x and g^y to be sent in the `ClientKeyShare` and `ServerKeyShare` messages exchanged between the tested and its (predicted) partnered session and uses the third DDH challenge value h as the ephemeral secret ES (as well as static secret SS) in both sessions.

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

Observe that, in case $h = g^{uv}$, this approach equals Game C.1 while, in case h is a random group element g^z for $z \leftarrow_{\$} \mathbb{Z}_q$, the simulation equals Game C.3.

We observe that $\mathcal{B}_6$ provides a perfect simulation for $\mathcal{A}$, both for $h = g^{uv}$ and for h being random. Most importantly, from $\mathcal{A}$'s perspective, g^u and g^v are chosen as in the real game since the (honest) tested session is contributively partnered (in stage 1) with another honest session (i.e., $\mathsf{label}.\mathsf{cid}_1 = \mathsf{label}'.\mathsf{cid}_1$) and thus the adversary cannot have modified the transcript between these two sessions (up to the derivation of the ephemeral secret in the tested session). Moreover, both values are chosen independently of the ephemeral values and temporary keys in all other sessions which the reduction can, hence, still select on its own. This in particular implies that an adversary, being able to notice the (coincidental) appearance of the same Diffie–Hellman key in another execution, still allows $\mathcal{B}_6$ to distinguish the two cases by $\mathcal{A}$'s differing behavior.

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

$$\mathsf{Adv}^{G_{C.1}}_{\texttt{draft-10-full},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.3}}_{\texttt{draft-10-full},\mathcal{A}} + \mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_6}.$$

Game C.4. In this game, we replace the extracted ephemeral and static secrets xES and xSS (which are equal as ES = SS) by a uniform and independent random string $\widetilde{\text{xES}} = \widetilde{\text{xSS}} \leftarrow_{\$} \{0,1\}^\lambda$ in the tested session and, if derived there, in the partnered session.
We can turn any adversary $\mathcal{A}$ able to distinguish this change with non-negligible probability into an adversary $\mathcal{B}_7$ against the security of the $\mathsf{HKDF.Extract}$ function which we model as a pseudorandom function (defined for keys chosen at random from $\mathbb{G}$). We let $\mathcal{B}_7$ simulate Game C.3 as the challenger, except that it uses its PRF oracle for the derivation of xES and xSS both in the tested and its partnered session. Observe that, in case the oracle computes the PRF function, this equals Game C.3, whereas, if it computes a random function, this equals Game C.4. The simulation is sound because the ephemeral and static secrets $\widetilde{\text{ES}} = \widetilde{\text{SS}}$, by the change in Game C.3, are a random element in $\mathbb{G}$ chosen independently of all other values in the game and the same salt value 0 is used for the derivation of xES and xSS (leading to xES = xSS always being true).

The advantage of $\mathcal{B}_7$ in the PRF security game therefore bounds the advantage difference such that

$$\mathsf{Adv}^{G_{C.3}}_{\texttt{draft-10-full},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.4}}_{\texttt{draft-10-full},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{HKDF.Extract},\mathcal{B}_7}.$$

Game C.5. Next, we replace the handshake traffic key tk_{hs}, the expanded ephemeral and static secrets mES and mSS, and the finished secret FS derived in both the tested and its partnered session by independent uniformly random values $\widetilde{tk_{hs}}, \widetilde{\text{mES}}, \widetilde{\text{mSS}}, \widetilde{\text{FS}} \leftarrow_{\$} \{0,1\}^\lambda$. Recall that in contrast to the extracted secrets xES and xSS that are derived using the same salt, the expanded secrets mES and mSS are computed using distinct labels.
Similar to the step in Game C.4, we can bound the difference in $\mathcal{A}$'s advantage introduced through this step by the security of the $\mathsf{HKDF.Expand}$ function which we model as a pseudorandom function, this time defined for keys being uniformly random bit strings from $\{0,1\}^\lambda$. The reduction $\mathcal{B}_8$, analogously to the previous step, uses its PRF oracle for the evaluations of $\mathsf{HKDF.Expand}$ under the key $\widetilde{\text{xES}} = \widetilde{\text{xSS}}$ in the tested and its partnered session. Depending on the oracles behavior it again perfectly simulates either Game C.4 or Game C.5, as $\widetilde{\text{xES}} = \widetilde{\text{xSS}}$ is a uniformly random and independent bit string.

We can hence can infer that

$$\mathsf{Adv}^{G_{C.4}}_{\texttt{draft-10-full},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.5}}_{\texttt{draft-10-full},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}.$$

Game C.6. As the second to last step, we replace the master secret MS by a uniformly random value $\widetilde{\mathrm{MS}}$. This again can be bounded by the advantage against the PRF security (with uniformly random keys) of $\mathsf{HKDF.Extract}$ as MS is derived from key $\widetilde{\mathrm{mES}}$ and salt mSS, now independent uniformly random bit strings. Therefore,

$$\mathsf{Adv}^{G_{C.5}}_{\texttt{draft-10-full},\mathcal{A}} \leq \mathsf{Adv}^{G_{C.6}}_{\texttt{draft-10-full},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_9}.$$

Game C.7. Finally, we replace all $\mathsf{HKDF.Expand}$ evaluations using the (replaced) master secret $\widetilde{\mathrm{MS}}$ as key in the tested and its partnered session by a (lazy-sampled) random function. This change affects the derivation of the handshake traffic key tk_{app}, the resumption master secret RMS, and the exporter master secret EMS which are hereby replaced with independent random values $\widetilde{tk_{app}}, \widetilde{\mathrm{RMS}}, \widetilde{\mathrm{EMS}} \leftarrow_{\$} \{0,1\}^{\lambda}$ in in both sessions.
As in the previous three steps, we can bound the difference in $\mathcal{A}$'s advantage introduced through this step by the PRF security of $\mathsf{HKDF.Expand}$, again defined for keys being uniformly random bit strings from $\{0,1\}^{\lambda}$. To this extent, the reduction $\mathcal{B}_{10}$ as above uses its PRF oracle for all evaluations of $\mathsf{HKDF.Expand}$ under the key $\widetilde{\mathrm{MS}}$ in the tested and its partnered session. Depending on the oracles behavior, this perfectly simulates either Game C.6 or Game C.7, as $\widetilde{\mathrm{MS}}$ is a uniformly random and independent bit string and different labels are used in the derivation of tk_{app}, RMS, and EMS.

We can hence can infer that

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

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

$$\mathsf{Adv}^{G_{C.7}}_{\texttt{draft-10-full},\mathcal{A}} \leq 0.$$

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

B Multi-Stage Preshared-Secret Key Exchange Model

We modify the multi-stage preshared-secret key exchange (MS-PSKE) model from our previous work to also cover the forward secrecy aspects of the PSK / PSK-(EC)DHE TLS 1.3 draft-10 handshake variants. Given that the first presentation of the MS-PSKE model was itself a high-level description of the changes between MSKE and MS-PSKE, we now give a full reproduction of the model below:

B.1 Preliminaries

We denote by $\mathcal{U}$ the set of *identities* used to model the participants in the system, identified by some $U \in \mathcal{U}$. Sessions of a protocol are identified as before using a *label* $\mathsf{label} \in \mathsf{LABELS} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$, where (U, V, k) indicates the k-th local session of identity U (the session *owner*) with V as the intended communication *partner*. Each session is also associated with a key index of the preshared secret pss used in the protocol run. Each pss is uniquely identified via a public preshared secret identifier psid. The challenger generates pss values when prompted by the adversary, maintaining two lists of $(n_u) \cdot (n_u - 1)$ vectors of: preshared secrets between distinct protocol participants, denoted $\mathsf{pss}_{\vec{U,V}}$, and preshared secret identifiers of the preshared secrets between distinct protocol participants, denoted $\vec{\mathsf{psid}}_{U,V}$. Note that the kth entry in $\vec{\mathsf{pss}}_{U,V}$ corresponds to the kth secret shared between parties U and V, and the kth entry in $\vec{\mathsf{psid}}_{U,V}$ corresponds to its preshared secret identifier. We follow MSKE in defining authentication types unauth and stage-j mutual authentication. Our secret compromise paradigm also follows MSKE in allowing leakage of long-term preshared secrets, and session keys, while disallowing leakage of ephemeral secrets and session state.

For each session, a tuple with the following information is maintained as an entry in the *session list* $\mathsf{List_S}$.

- $\mathsf{label} \in \mathsf{LABELS}$: the (administrative) session label
- $U \in \mathcal{U}$: the session owner
- $V \in (\mathcal{U} \cup \{*\})$: the intended communication partner, allowing the identity of the communication partner to be set once during the protocol run
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: the session owner's role in this session
- $\mathsf{auth} \in \mathsf{AUTH} \subseteq \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}^{\mathsf{M}}$: the aspired authentication type of each stage from the set of supported properties AUTH
- $k \in \mathbb{N}$: the index of the preshared secret used in a protocol run with the communication partner
- $\mathsf{pss} \in (\{0,1\}^* \cup \{\perp\})$: the preshared secret to be used in the session
- $\mathsf{psid} \in (\{0,1\}^* \cup \{\perp\})$: the preshared secret identifier of the preshared secret to be used in the session
- $\mathsf{st_{exec}} \in (\mathsf{RUNNING} \cup \mathsf{ACCEPTED} \cup \mathsf{REJECTED})$: the state of execution $[\mathsf{running_0}]$
- $\mathsf{stage} \in \{0, \ldots, \mathsf{M}\}$: the current stage [0], where stage is incremented to i when $\mathsf{st_{exec}}$ reaches $\mathsf{accepted}_i$ resp. $\mathsf{rejected}_i$
- $\mathsf{sid} \in (\{0,1\}^* \cup \{\perp\})^{\mathsf{M}}$: sid_i $[\perp]$ indicates the session identifier in stage $i > 0$
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\perp\})^{\mathsf{M}}$: cid_i $[\perp]$ indicates the contributive identifier in stage $i > 0$
- $\mathsf{K} \in (\{0,1\}^* \cup \{\perp\})^{\mathsf{M}}$: K_i $[\perp]$ indicates the established session key in stage $i > 0$
- $\mathsf{st_{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}^{\mathsf{M}}$: $\mathsf{st}_{\mathsf{key},i}$ $[\mathsf{fresh}]$ indicates the state of the session key in stage $i > 0$
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^{\mathsf{M}}$: test indicator tested_i $[\mathsf{false}]$, where true means that K_i has been tested

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

B.2 Adversary Model

We restate the differences between MS-PSKE and MSKE. Note that the Send, Test and Reveal queries are virtually verbatim. The adversary can interact with the protocol via the following queries:

- $\mathsf{NewSecret}(U, V, k, \mathsf{psid})$: The challenger first checks that $\vec{\mathsf{pss}}_{U,V}$ does not already have an entry at k, returning $\perp$ if so. The challenger also checks that psid does not already exist in the experiment as a psid for any pss, returning $\perp$ to the adversary if so. This ensures global uniqueness of the psid value. Creates a new preshared secret sampled uniformly at random and independently of each other from preshared secret space and stores it as the kth entry of $\vec{\mathsf{pss}}_{U,V}$ and $\vec{\mathsf{pss}}_{V,U}$. Also stores the adversary-provided psid value as the *kth* entry of $\vec{\mathsf{psid}}_{U,V}$ and $\vec{\mathsf{psid}}_{V,U}$.

- $\mathsf{NewSession}(U, V, k, \mathsf{role}, \mathsf{auth})$: Creates a new session for participant identity U with role role and aiming at authentication type auth. If the challenger has not yet generated a preshared secret between U and V with key index k, return $\perp$. Otherwise the challenger generates a unique new label label and adds the entry $(\mathsf{label}, U, V, k, \mathsf{role}, \mathsf{auth})$ to $\mathsf{List_S}$. The challenger sets the per-session variable $\mathsf{label.psid}$ to the kth entry of $\vec{\mathsf{psid}}_{U,V}$, and the per-session variable $\mathsf{label.pss}$ to the kth entry of $\vec{\mathsf{pss}}_{U,V}$.

- $\mathsf{Corrupt}(\mathsf{label})$: Provides the session preshared secret $\mathsf{label.pss}$ to the adversary. No other queries are allowed to sessions with labels label' such that $\mathsf{label}'.\mathsf{psid} = \mathsf{label.psid}$. In the non-forward secret case, for each session with label' such that $\mathsf{label}'.\mathsf{psid} = \mathsf{label}.\mathsf{psid}$, set $\mathsf{label}'.\mathsf{st}_{\mathsf{key},i}$ (for all i) to $\mathsf{revealed}$. All keys output by sessions with the same preshared secret are considered disclosed. In the case of stage-j forward secrecy, for each session with label' such that $\mathsf{label}'.\mathsf{psid} = \mathsf{label}.\mathsf{psid}$, set $\mathsf{label}'.\mathsf{st}_{\mathsf{key},i}$ to $\mathsf{revealed}$ only if $i < j$ or $i > \mathsf{stage}$. This captures the notion that previous stage keys that are forward secret are not considered disclosed. Reveal queries issued to sessions as a result of key-dependent security are processed as in the $\mathsf{Corrupt}$ query definition of the MSKE model.

- $\mathsf{Send}(\mathsf{label}, m)$: Sends a message m to the session with label label. If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, k, \mathsf{psid}, \mathsf{pss}, \mathsf{st}_{\mathsf{exec}}, \mathsf{stage}, \mathsf{sid}, \mathsf{cid}, \mathsf{K}, \mathsf{st}_{\mathsf{key}}, \mathsf{tested})$ in $\mathsf{List}_{\mathsf{S}}$, return $\perp$. Otherwise, run the protocol on behalf of U on message m and return the response and the updated state of execution $\mathsf{st}_{\mathsf{exec}}$. As before, if $\mathsf{role} = \mathsf{initiator}$ and $m = \mathsf{init}$, the protocol is initiated. If during protocol execution, the state of execution changes to $\mathsf{accepted}_i$ for some i, the protocol execution is immediately suspended and $\mathsf{accepted}_i$ is returned as result to the adversary, and can later trigger the resumption of the protocol execution by issuing a special $\mathsf{Send}(\mathsf{label}, \mathsf{continue})$ query. This is to allow the adversary to test a session key before use in later stages prevents it. If the state of execution changes to $\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$ for some i and there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', k, \mathsf{psid}', \mathsf{pss}', \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{st}'_{\mathsf{key},i} = \mathsf{revealed}$, then, for key-independence, $\mathsf{st}_{\mathsf{key},i}$ is set to $\mathsf{revealed}$ as well, whereas for key-dependent security, all $\mathsf{st}_{\mathsf{key},i'}$ for $i' \geq i$ are set to $\mathsf{revealed}$. If the state of execution changes to $\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$ for some i and there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', k, \mathsf{psid}', \mathsf{pss}', \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{tested}'_i = \mathsf{true}$, then set $\mathsf{label}.\mathsf{K}_i \leftarrow \mathsf{label}'.\mathsf{K}'_i$ and $\mathsf{label}.\mathsf{tested}_i \leftarrow \mathsf{true}$. If the state of execution changes to $\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$ for some i and the intended communication partner V is corrupted, then set $\mathsf{st}_{\mathsf{key},i} \leftarrow \mathsf{revealed}$.

- $\mathsf{Reveal}(\mathsf{label}, i)$: Reveals $\mathsf{label}.\mathsf{K}_i$, the session key of stage i in the session with label label. If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, k, \mathsf{psid}, \mathsf{pss}, \mathsf{st}_{\mathsf{exec}}, \mathsf{stage}, \mathsf{sid}, \mathsf{cid}, \mathsf{K}, \mathsf{st}_{\mathsf{key}}, \mathsf{tested})$ in $\mathsf{List}_{\mathsf{S}}$, or $i > \mathsf{stage}$, or $\mathsf{tested}_i = \mathsf{true}$, then return $\perp$. Otherwise, set $\mathsf{st}_{\mathsf{key},i}$ to $\mathsf{revealed}$ and provide the adversary with K_i. If there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', k, \mathsf{psid}', \mathsf{pss}', \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{stage}' \geq i$, then $\mathsf{st}'_{\mathsf{key},i}$ is set to $\mathsf{revealed}$ to ensure that partnered session keys are also considered revealed. If $i = \mathsf{stage}$, set $\mathsf{st}_{\mathsf{key},j} = \mathsf{revealed}$ for all $j > i$, as they may depend on the revealed key. If a partnered session label' with $\mathsf{sid}_i = \mathsf{sid}'_i$ has $\mathsf{stage}' = i$, then set $\mathsf{st}'_{\mathsf{key},j} = \mathsf{revealed}$ for all $j > i$. Note that if however $\mathsf{stage}' > i$, then keys K'_j for $j > i$ derived in the partnered session are not considered to be revealed by this query since they have been accepted previously, i.e., prior to K_i being revealed in this query.

- $\mathsf{Test}(\mathsf{label}, i)$: Tests the session key of stage i in the session with label label. In the security game this oracle is given a uniformly random test bit b_{test} as state which is fixed throughout the game. If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{auth}, k, \mathsf{psid}, \mathsf{pss}, \mathsf{st}_{\mathsf{exec}}, \mathsf{stage}, \mathsf{sid}, \mathsf{cid}, \mathsf{K}, \mathsf{st}_{\mathsf{key}}, \mathsf{tested})$ in $\mathsf{List}_{\mathsf{S}}$ or if $\mathsf{label}.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{accepted}_i$, return $\perp$. If there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', k, \mathsf{psid}', \mathsf{pss}', \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$, but $\mathsf{st}'_{\mathsf{exec}} \neq \mathsf{accepted}_i$, set the 'lost' flag to $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that keys can only be tested if they have just been accepted but not used yet, including ensuring any partnered session that may have already established this key has not used it. If $\mathsf{label}.\mathsf{auth}_i = \mathsf{unauth}$, but there is no tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', k, \mathsf{psid}', \mathsf{pss}', \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ (for $\mathsf{label} \neq \mathsf{label}'$) in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{cid}_i = \mathsf{cid}'_i$, then set $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that having an honest contributive partner is a prerequisite for testing responder sessions in an unauthenticated or unilaterally authenticated stage and for testing an initiator session in an unauthenticated stage.[7] If $\mathsf{label}.\mathsf{tested}_i = \mathsf{true}$, return K_i, ensuring that repeated queries will be

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

answered consistently. Otherwise, set $\mathsf{label.tested}_i$ to true. If the test bit b_{test} is 0, sample $\mathsf{label.K}_i \leftarrow_{\$} \mathcal{D}$ at random from the session key distribution $\mathcal{D}$. This means that we substitute the session key by a random and independent key which is also used for future deployments *within* the key exchange protocol. Moreover, if there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{auth}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{cid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$, also set $\mathsf{label}'.\mathsf{K}'_i \leftarrow \mathsf{label.K}_i$ and $\mathsf{label}'.\mathsf{tested}'_i \leftarrow \mathsf{true}$ to ensure consistency in the special case that both label and label' are in state $\mathsf{accepted}_i$ and, hence, either of them can be tested first.
Return $\mathsf{label.K}_i$.

B.3 Security of Multi-Stage Preshared Key Exchange Protocols

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

B.3.1 Match Security

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

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

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

Definition B.1 (Match security)**.** *Let* KE *be a key exchange protocol and $\mathcal{A}$ a PPT adversary interacting with* KE *via the queries defined in Section B.2 in the following game $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$:*
Query. *The adversary $\mathcal{A}$ has access to the queries* NewSecret*,* NewSession*,* Send*,* Reveal*, and* Corrupt*.*
Stop. *At some point, the adversary stops with no output.*
We say that $\mathcal{A}$ wins the game, denoted by $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1$, if at least one of the following conditions hold:

1. *There exist two distinct labels* label, label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, $\mathsf{label.st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$, $\mathsf{label}'.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$, *but* $\mathsf{label.K}_i \neq \mathsf{label}'.\mathsf{K}_i$. *(Different session keys in the same stage of partnered sessions.)*
2. *There exist two distinct labels* label, label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *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.)*
3. *There exist two distinct labels* label, label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, *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.)*
4. *There exist two distinct labels* label, label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, $\mathsf{label.auth}_i = \mathsf{label}'.\mathsf{auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$, $\mathsf{label.role} = \mathsf{initiator}$, *and* $\mathsf{label}'.\mathsf{role} = \mathsf{responder}$, *but* $\mathsf{label}.V \neq \mathsf{label}'.U$ *or (only if* $\mathsf{label.auth}_i = \mathsf{mutual}$*)* $\mathsf{label}.U \neq \mathsf{label}'.V$ *or (only if* $\mathsf{label.auth}_i = \mathsf{mutual}$*)* $\mathsf{label}.k \neq \mathsf{label}'.k$.

5. *There exist two (not necessarily distinct) labels* label, label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_j \neq \bot$ *for some stages* $i, j \in \{1, \ldots, \mathsf{M}\}$ *with* $i \neq j$. *(Different stages share the same session identifier.)*
6. *There exist three distinct labels* label, label', label'' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i = \mathsf{label}''.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$. *(More than two sessions share the same session identifier.)*

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

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

B.3.2 Multi-Stage Security

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

Definition B.2 (Multi-Stage security)**.** *Let* KE *be a preshared key exchange protocol with (session) key distribution* $\mathcal{D}$, *and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section B.2 in the following game* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$:

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

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

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

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

We say that $\mathcal{A}$ *wins the game, denoted by* $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1$, *if* $b = b_{\mathsf{test}}$ *and* $\mathsf{lost} = \mathsf{false}$. *We say* KE *is* Multi-Stage-secure *in a key-dependent resp. key-independent and non-forward-secret resp. stage-j forward-secret manner with concurrent authentication properties* AUTH *if* KE *is* Match*-secure and for all PPT adversaries* $\mathcal{A}$ *the following advantage is negligible in the security parameter:*

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

C Security Analysis of the TLS 1.3 draft-10 Pre-shared Key Handshakes

C.1 Technical Specification for `draft-10-PSK(EC)DHE` and `draft-10-PSK` in the Multi-Stage Key Exchange Model

We begin by defining the session identifiers for the two stages (note that RMS and EMS are not computed in the PSK or PSK-(EC)DHE handshakes), deriving the handshake traffic key tk_{hs} and the application traffic key tk_{app} to be be (as in `draft-ietf-tls-tls13-10` analysis) the unencrypted messages sent and received excluding the finished message:

$$\begin{aligned}
\mathsf{sid}_1 = (&\mathtt{ClientHello}, \mathtt{ClientKeyShare}, \mathtt{ClientPreSharedKey},\\
&\mathtt{ServerHello}, \mathtt{ServerKeyShare}^{\dagger}, \mathtt{ServerPreSharedKey})\\
\mathsf{sid}_2 = (&\mathtt{ClientHello}, \mathtt{ClientKeyShare}, \mathtt{ClientPreSharedKey},\\
&\mathtt{ServerHello}, \mathtt{ServerKeyShare}^{\dagger}, \mathtt{ServerPreSharedKey}, \mathtt{EncryptedExtensions}, \text{“}tk_{app}\text{”})\\
\mathsf{sid}_3 = (&\mathsf{sid}_2, \text{“EMS”}).
\end{aligned}$$

Note that † indicates messages only included in the PSK-(EC)DHE handshake mode.

We add flags to the session identifiers to ensure session identifiers for each stage are distinct. The contributive identifiers are incrementally set with each flow of messages sent and received by each party. So $\mathsf{cid}_1 = (\texttt{ClientHello}, \texttt{ClientKeyShare}, \texttt{ClientPreSharedKey})$ extended to $\mathsf{cid}_1 = \mathsf{sid}_1$ and $\mathsf{cid}_2 = \mathsf{sid}_2$, $\mathsf{cid}_3 = \mathsf{sid}_3$, similarly to how `draft-ietf-tls-tls13-10` incrementally sets cid_1, cid_2, and cid_3.

C.2 Proof of draft-10-PSK(EC)DHE Match Security

We need to show the six properties of Match security. Note that the only condition that is changed between Multi-Stage Key Exchange (MSKE) model and Multi-Stage Pre-Shared Key Exchange (MS-PSKE) is the fourth one, which now also requires agreement on the preshared-secret identifier psid. In addition, there are minor changes to account for the preshared secret, as well as the fact that RMS is not output.

1. *Sessions with the same session identifier for some stage hold the same session key.* As before, since the session identifier contains the parties' Diffie–Hellman contributions g^x and g^y which uniquely identify the Diffie–Hellman key, as well as all data entering the key derivation step and the preshared-secret identifier psid. Hence, equal session identifiers imply that both parties compute the same ephemeral secret and thus same handshake traffic key tk_{hs} on the first stage. For the second stage note that the identifier sid_2 contains the full sid_1, implying that the parties have also computed the same ephemeral secret. Since the key derivation for the application traffic key tk_{app} is only based on this secret value and data from sid_2 and pss, it follows that the session keys must be equal, too.
2. *Sessions with the same session identifier for some stage agree on the authenticity of the stage.* Since draft-10-PSK(EC)DHE only specifies (unauth, mutual), this is trivally true.
3. *Sessions with the same session identifier for some stage share the same contributive identifier.* This holds again since the contributive identifier values cid_1, cid_2, cid_3 are final and equal to the respective session identifiers once the session identifiers sid_1, sid_2, sid_3 are set.
4. *Sessions are partnered with the intended partner.* Honest sessions are assured of a peer's identity and key index as the preshared secret identifier psid is included in sid_1 and sid_2 as `ClientPreSharedKey` and `ServerPreSharedKey`. Since each party knows the mapping of key index k and psid, a party can determine peer identity via this mapping, and agreement on sid_1, sid_2 implies agreement on partner identity.
5. *Session identifiers are distinct for different stages.* This holds as sid_2 and sid_3 contain distinct labels ("tk_{app}" resp. "EMS") that are not contained in sid_1.
6. *At most two sessions have the same session identifier at any stage.* Both client and server nonces and g^x, g^y are included in both stages session identifiers sid_1, sid_2, and thus the probability of three-fold colliding session identifiers is bound by the probability of both nonce collisions and thus: $n_s^2 \cdot 1/q \cdot 2^{-|nonce|}$ where n_s is the maximum number of sessions, q is the group order, and $|nonce| = 128$ the nonces' bit-length.

□

C.3 Proof of draft-10-PSK(EC)DHE Multi-Stage Security

As in our previous proofs, we consider the case that the adversary $\mathcal{A}$ makes a single Test query, reducing the advantage of $\mathcal{A}$ by a factor of $1/3n_s$ (as RMS is not computed in the pre-shared modes of `draft-10`). Additionally, we now know the session with label label that is to be tested in stage i. Our analysis considers two disjoint cases:

A. The adversary tests a session without honest contributive partner in the first stage[8]

[8] Note that this tested session can either have an initiator or responder role.

B. The adversary tests a session with an honest contributive partner in the first stage

Case A. Test Session without Partner

We first consider the case that the tested session is without honest contributive partner in the first stage. Since for `draft-10-PSK(EC)DHE` the first stage is always unauthenticated, the adversary cannot test a session in the first stage without an honest contributive partner, this restricts our focus to Test queries in stage 2 and 3. We proceed in the following sequence of game hops, where each game iteratively changes the original Multi-Stage game and bound the advantage difference of adversary $\mathcal{A}$ between any two games by complexity-theoretic assumptions.

Game A.0. This initial game equals the Multi-Stage game with a single Test query issued for a stage 2 session without honest contributive partner in stage 1. Thus,

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} = \mathsf{Adv}^{\text{1-Multi-Stage,session without partner}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}}.$$

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

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

$$\mathsf{Adv}^{G_{A.0}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

Game A.2. In this game, the challenger aborts immediately if a session accepts in the second stage without an honest contributive partner in stage 1. Let $\mathsf{abort}^{G_{A.2},\mathcal{A}}_{acc}$ denote this event this occurs. Then

$$\left|\mathsf{Adv}^{G_{A.1}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} - \mathsf{Adv}^{G_{A.2}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}}\right| \leq \Pr[\mathsf{abort}^{G_{A.2},\mathcal{A}}_{acc}].$$

We can immediately bound $\mathsf{Adv}^{G_{A.2}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}}$. Throughout this proof case we assume that the Test query is directed to a session without honest contributed partner in stage 1. Because the authentication type of the protocol is $(\mathsf{unauth}, \mathsf{mutual}, \mathsf{mutual})$, the Test query can only be directed to stage-2 or stage-3 keys of that session. As Game A.2 is aborted when the first such session accepts (in stage 2), there is after all no moment in that game where a successful adversary could issue a Test query. Hence,

$$\mathsf{Adv}^{G_{A.2}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} = 0.$$

It remains to bound $\Pr[\mathsf{abort}^{G_{A.2},\mathcal{A}}_{acc}]$. We do so via a sequence of games that continues on from Game A.2.

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 session which accepts in the second stage without an honest contributive partner in stage 1. 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:

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

Note that, in this game, the guessed session, which is the first stage-2 session that accepts without honest contributive partner in the first stage, could not have been issued any $\mathsf{Corrupt}$ query, nor could a $\mathsf{Corrupt}$ query have been issued to any other session sharing the same pre-shared secret. This is because sessions using that pre-shared secret do not continue execution once the secret is corrupted, and this session has accepted, so no $\mathsf{Corrupt}$ could have happened before it accepted in stage 2. Since the game terminates once stage 2 has accepted, no $\mathsf{Corrupt}$ query could have been issued after, either.

This allows us, in the following games, to replace the pre-shared secret pss in the guessed and all other sessions sharing the same pss value without being inconsistent or detectable with regards to the $\mathsf{Corrupt}$ query.

Game A.4. In this game we guess the pre-shared secret pss that the guessed session will use, and the challenger aborts the game if that guess was wrong. This reduces the adversary's advantage by a factor of at most $1/n_s$, thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.3},\mathcal{A}}] \leq n_s \cdot \Pr[\mathsf{abort}_{acc}^{G_{A.4},\mathcal{A}}].$$

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

Game A.5. We next replace the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations using the guessed session's pre-shared secret $\mathsf{pss}_{U,V,k}$ as key by a lazy-sampled random function. Beyond other sessions using the same pre-shared secret, this in particular affects the derivation of xSS in the guessed session, which is replaced with a random value $\widetilde{\mathrm{xSS}} \leftarrow_{\$} \{0,1\}^{\lambda}$.
We bound this difference of the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Extract}$. In the case of the oracle computing the function, the simulation equals Game A.4, but if it computes a random function, the simulation equals Game A.5. For any successful adversary (note that a successful adversary by Games A.3 and A.4 cannot corrupt $\mathsf{pss}_{U,V,k}$, i.e., cannot issue $\mathsf{Corrupt}(\mathsf{label})$ queries where $\mathsf{label.pss} = \mathsf{pss}_{U,V,k}$) the pre-shared secret is uniformly random and unknown to $\mathcal{A}$, so the simulation is sound. Thus

$$\Pr[\mathsf{abort}_{acc}^{G_{A.4},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{A.5},\mathcal{A}}] + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2}.$$

Game A.6. In this step we replace the evaluations of $\mathsf{HKDF.Expand}$ using $\widetilde{\mathrm{xSS}}$ as key in the guessed session by a lazy-sampled random function, thereby exchanging the finished secret value FS, and the expanded static secret mSS with independent random values $\widetilde{\mathrm{FS}}$, $\widetilde{\mathrm{mSS}}$. We can bound this difference in the same manner as above, and thus:

$$\Pr[\mathsf{abort}_{acc}^{G_{A.5},\mathcal{A}}] \leq \Pr[\mathsf{abort}_{acc}^{G_{A.6},\mathcal{A}}] + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3}.$$

Finally, we show how any adversary that manages to make the $\mathsf{abort}_{acc}^{G_{A.6},\mathcal{A}}$ event happen can be transformed into an adversary $\mathcal{B}_4$ that breaks the existential unforgeability of the HMAC scheme.

To this extent, let $\mathcal{B}_4$ simulate Game A.6 for $\mathcal{A}$ as specified, but when the guessed session or partner session requires a MAC computation using $\widetilde{\mathrm{FS}}$, $\mathcal{B}_4$ invokes its MAC oracle to generate that value. Since $\widetilde{\mathrm{FS}}$ is uniformly random and independent of all other values in the game, this simulation is sound.

Assume now $\mathcal{A}$ triggers $\mathsf{abort}_{acc}^{G_{A.6},\mathcal{A}}$. In this case, the accepting session must have received a $\mathtt{SF}$ (respectively, $\mathtt{CF}$) message (when $\mathsf{role} = \mathsf{initiator}$, resp. $\mathsf{responder}$) that is a valid MAC tag over the session hash $\mathsf{H}(\mathtt{CH}, ..., \mathtt{EE})$. Since every other honest session holds a different session identifier (as there exists no honest contributive partner in the first stage of the accepting session), no honest party will have issued a MAC tag on that session hash. Moreover, there exist no hash collisions by Game A.1, so the MAC input is distinct to any other MAC input for any honest party. Therefore, this message was never queried to the

MAC oracle and hence constitutes a MAC forgery. This allows us to conclusively bound the probability for abortion due to a stage-2 accepting session without stage-1 contributive identifier by

$$\Pr[\mathsf{abort}_{acc}^{G_{A.6},\mathcal{A}}] \le \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_4}.$$

Summing the probabilities accumulated over the sequence of games, we obtain the result:

$$\mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{session without partner}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \le \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_s^2 \cdot \left(\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_4}\right).$$

Case B. Test Session with Partner

We now come to the case where the tested session has an honest contributive partner in the first stage.

Game B.0. This initial game equals the Multi-Stage game with a single Test query issued for a session with an honest contributive partner in stage 1. Thus,

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} = \mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{session with partner}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}}.$$

Game B.1. In this game, we let the challenger abort the game if any two honest sessions compute the same hash value for different inputs in any evaluation of the hash function H.
Let $\mathsf{abort}_{\mathsf{H}}$ denote the event that the challenger aborts in this case. We can bound the probability $\Pr[\mathsf{abort}_{\mathsf{H}}]$ by the advantage $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5}$ of an adversary $\mathcal{B}_5$ against the collision resistance of the hash function H. To this extent, $\mathcal{B}_5$ acts as the challenger in Game B.1, using its description of H to compute hash values, and running adversary $\mathcal{A}$ as a subroutine. If the event $\mathsf{abort}_{\mathsf{H}}$ occurs, $\mathcal{B}_5$ outputs the two distinct input values to H resulting in the same hash value as a collision.

Note that $\mathcal{B}_5$ perfectly emulates the attack of $\mathcal{A}$ according to $G_{B.2}$ up to the point when a hash collision occurs. As $\mathcal{B}_5$ wins if $\mathsf{abort}_{\mathsf{H}}$ is triggered, we have that $\Pr[\mathsf{abort}_{\mathsf{H}}] \le \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5}$ and thus:

$$\mathsf{Adv}^{G_{B.0}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \le \mathsf{Adv}^{G_{B.1}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5}.$$

Game B.2. Our second modification is to 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 1 of the tested session. This reduces the adversary's advantage by a factor of at most $1/n_s$.

$$\mathsf{Adv}^{G_{B.1}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \le n_s \cdot \mathsf{Adv}^{G_{B.2}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}}.$$

Game B.3. In this game, we replace the ephemeral secret ES computed in both the tested session and its contributive partner session by a randomly chosen group element $\widetilde{\mathrm{ES}} = g^z$ for $z \leftarrow_{\$} \mathbb{Z}_q$. If $\mathcal{A}$ is able to distinguish this change, we can create an adversary $\mathcal{B}_6$ against the decisional Diffie-Hellman (DDH) assumption. In simulating the game, $\mathcal{B}_6$ uses the challenge group elements g^u, g^v and h from the DDH challenger, in the `ClientKeyShare`, `ServerKeyShare` messages and the derived secret ES respectively. If $h = g^{uv}$, this simulation is equal to Game B.1, but if h is a random element, then the simulation equal to Game B.3. Since we are in Case B, the tested session must have a contributive partner session in stage 1, and thus the simulation is sound. Note that this is due to cid_1 containing both DH shares, and (by assumption) the tested session has an honest contributive partner in the first stage (i.e. both session's cid_1 values are equal) and the adversary cannot modify these values. Both values are chosen independently of all other values chosen in the game. Thus:

$$\mathsf{Adv}^{G_{B.2}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \le \mathsf{Adv}^{G_{B.3}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_6}.$$

Game B.4. In this game, we replace the extracted ephemeral secret xES derived in both the tested and its contributive partner session with a uniformly random and independent string $\widetilde{\text{xES}} \leftarrow_\$ \{0,1\}^\lambda$ in the tested and potentially partner session. We can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_7$ against the security of the $\mathsf{HKDF.Extract}$ function which we still model as a pseudorandom function with keys from $\mathbb{G}$. We let $\mathcal{B}_7$ simulate the previous game as the challenger, except it queries its PRF oracle for the derivation of xES in both the tested and partner session. If the oracle computes the PRF, we are in Game B.3, but if it computes a random function, we are in Game B.4 as ES is uniformly random and independent value. The advantage of $\mathcal{B}_7$ in the PRF security game bounds the advantage of this change, such that:

$$\mathsf{Adv}^{G_{B.3}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.4}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{HKDF.Extract},\mathcal{B}_7}.$$

Game B.5. In this game, we replace the handshake traffic key tk_{hs} and the expanded ephemeral secret mES derived in both the tested and its contributive partner session with a uniformly random and independent strings $\widetilde{tk_{hs}}$, $\widetilde{\text{mES}} \leftarrow_\$ \{0,1\}^\lambda$ in the tested and partner session. We can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_8$ against the security of the $\mathsf{HKDF.Expand}$ function which we still model as a pseudorandom function. By a similar change to Game B.4, we have:

$$\mathsf{Adv}^{G_{B.4}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.5}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}.$$

Game B.6. In this game, we replace the master secret MS derived from $\widetilde{\text{mES}}$ in both the tested and its contributive partner session with a uniformly random and independent string $\widetilde{\text{MS}} \leftarrow_\$ \{0,1\}^\lambda$ in the tested and partner session. We can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_9$ against the security of the $\mathsf{HKDF.Extract}$ function which we still model as a pseudorandom function. Via a similar argument to the previous games we find:

$$\mathsf{Adv}^{G_{B.5}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.6}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_9}.$$

Game B.7. In this game, we replace the application traffic key tk_{app} and the exporter master secret EMS derived from $\widetilde{\text{MS}}$ in both the tested and its contributive partner session with a uniformly random and independent strings $\widetilde{tk_{app}}$, $\widetilde{\text{EMS}} \leftarrow_\$ \{0,1\}^\lambda$ in the tested and partner session. We can turn any adversary capable of distinguishing this change into an adversary $\mathcal{B}_{10}$ against the security of the $\mathsf{HKDF.Expand}$ function which we still model as a pseudorandom function. Via a similar argument as the previous games we find:

$$\mathsf{Adv}^{G_{B.6}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{G_{B.7}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}}.$$

Note that now $\widetilde{tk_{hs}}$, $\widetilde{tk_{app}}$ and $\widetilde{\text{EMS}}$ are uniformly random bit strings independent of all other values. In particular the response to the Test query is now independent of the test bit b_{test}, and $\mathcal{A}$ cannot distinguish real from random case and thus:

$$\mathsf{Adv}^{G_{B.7}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \leq 0.$$

This yields the following security bound:

$$\mathsf{Adv}^{\mathsf{1\text{-}Multi\text{-}Stage},\text{session with partner}}_{\texttt{draft-10-PSK(EC)DHE},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_5} + n_s \cdot \Big(\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec},\mathbb{G}}_{\mathsf{HKDF.Extract},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_9} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}}\Big).$$

□

C.4 Proof of `draft-10-PSK` Multi-Stage Security

We again restrict the adversary $\mathcal{A}$ to make only a single Test query, reducing its advantage by a factor at most $1/3n_s$ via a hybrid argument that also fixes the tested session label and stage i.

Game 0. This initial game equals the $\mathsf{Multi\text{-}Stage}$ game with a single Test query, so

$$\mathsf{Adv}^{G_0}_{\texttt{draft-10-PSK},\mathcal{A}} = \mathsf{Adv}^{\text{1-Multi-Stage}}_{\texttt{draft-10-PSK},\mathcal{A}}.$$

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

$$\mathsf{Adv}^{G_0}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_1}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}.$$

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

$$\mathsf{Adv}^{G_1}_{\texttt{draft-10-PSK},\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{G_2}_{\texttt{draft-10-PSK},\mathcal{A}}.$$

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

Game 3. We next replace the pseudorandom function $\mathsf{HKDF.Extract}$ in all evaluations using the tested session's pre-shared secret $\mathsf{pss}_{U,V,k}$ as key by a (lazy-sampled) random function. This in particular affects the derivation of both the extracted ephemeral static secrets $\mathrm{xES} = \mathrm{xSS}$ in the tested (and a potential partnered) session, which is replaced by a random value $\widetilde{\mathrm{xES}} = \widetilde{\mathrm{xSS}} \leftarrow_\$ \{0,1\}^\lambda$.
We can bound the difference this step introduces in the advantage of $\mathcal{A}$ by the security of the pseudorandom function $\mathsf{HKDF.Extract}$. Notice here that for any successful adversary (which hence cannot invoke $\mathsf{Corrupt}$ on $\mathsf{pss}_{U,V,k}$ used in the tested session), the pre-shared key is an unknown and uniformly random value and, hence, the simulation is sound and we establish:

$$\mathsf{Adv}^{G_2}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_3}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_2}.$$

Game 4. We can now replace the $\mathsf{HKDF.Expand}$ applications in the tested and other sessions running on the same pre-shared key (and, hence, same ES and SS values) with a random function. Thereby, we in particular replace the handshake traffic key tk_{hs}, the expanded ephemeral and static secrecy mES, mSS, and the finished secret FS by random values $\widetilde{tk_{hs}}, \widetilde{\mathrm{mES}}, \widetilde{\mathrm{mSS}}, \widetilde{\mathrm{FS}} \leftarrow_\$ \{0,1\}^\lambda$. These values are moreover independent of any other values in the game, as the key derivation includes (a hash covering) the session identifier: Match security ensures no third session holds the same session identifier as the tested (and partnered) session and, due to Game 1, no hash collisions allows a different session identifier to be mapped to the same hash value.
We can, as before, bound the advantage difference introduces by this step by the PRF security of $\mathsf{HKDF.Expand}$ and obtain:

$$\mathsf{Adv}^{G_3}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_4}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_3}.$$

Game 5. Randomness and independence of $\widetilde{\mathrm{mES}}$ in the tested then allows us to replace the derived master secret by a random value $\widetilde{\mathrm{MS}}$, a step which is again reducible to the PRF security of $\mathsf{HKDF.Extract}$:

$$\mathsf{Adv}^{G_4}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_5}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_4}.$$

Game 6. As the last change, we can now replace the application traffic key tk_{app} and the exporter master secret EMS derived from $\widetilde{MS}$ by random values, which is undetectable given the PRF security of $\mathsf{HKDF.Expand}$:

$$\mathsf{Adv}^{G_5}_{\texttt{draft-10-PSK},\mathcal{A}} \leq \mathsf{Adv}^{G_6}_{\texttt{draft-10-PSK},\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}.$$

Finally, in Game 6 we reached a situation where all stages' keys in the tested session are chosen uniformly at random, which leaves $\mathcal{A}$ with no better chance then guessing:

$$\mathsf{Adv}^{G_6}_{\texttt{draft-10-PSK},\mathcal{A}} \leq 0.$$

Combining the given single bounds yields the overall security statement.