2017 IEEE European Symposium on Security and Privacy

Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates

Marc Fischlin Felix Günther
Cryptoplexity, Technische Universität Darmstadt
Darmstadt, Germany
marc.fischlin@cryptoplexity.de, guenther@cs.tu-darmstadt.de

***Abstract*—We investigate security of key exchange protocols supporting so-called zero round-trip time (0-RTT), enabling a client to establish a fresh provisional key without interaction, based only on cryptographic material obtained in previous connections. This key can then be already used to protect early application data, transmitted to the server before both parties interact further to switch to fully secure keys. Two recent prominent examples supporting such 0-RTT modes are Google's QUIC protocol and the latest drafts for the upcoming TLS version 1.3.**

We are especially interested in the question how replay attacks, enabled through the lack of contribution from the server, affect security in the 0-RTT case. Whereas the first proposal of QUIC uses state on the server side to thwart such attacks, the latest version of QUIC and TLS 1.3 rather accept them as inevitable. We analyze what this means for the key secrecy of both the preshared-key-based 0-RTT handshake in draft-14 of TLS 1.3 as well as the Diffie–Hellman-based 0-RTT handshake in TLS 1.3 draft-12. As part of this we extend previous security models to capture such cases, also shedding light on the limitations and options for 0-RTT security under replay attacks.

1. Introduction

Key exchange protocols are among the most widely used cryptographic protocols today, incorporated, e.g., in the TLS, SSH, IPsec, and QUIC protocols. They serve the purpose of establishing a (potentially authenticated) secret key between two parties in a network. While efficiency has always been a relevant aspect for such protocols, optimization traditionally focused on the cryptographic operations, which for a long time dominated the overall cost (in time) for executions. With the technological progress in speed of computation, but also advances and, equally important, the deployment of elliptic-curve cryptography, researchers and practitioners managed to reduce the cost of (even asymmetric) cryptographic operations drastically over the last decades. As a result, the communication complexity has become a more and more dominant factor for the overall efficiency of key exchange protocols.

1.1. Zero Round-Trip Time

While steadily increasing bandwidth on the Internet renders the data complexity aspect of communication subordinate, speed of light prepares to set a definitive lower bound for the time a message needs to be sent back and forth between two parties (called *round-trip time*). Reducing the round complexity has hence become a major design criteria in the last years, with several low-latency designs for key exchange proposed by researchers [29], [24], [16], [39] as well as by practitioners. Prominent practical examples are in particular Google's QUIC protocol [30] incorporated into the Chrome browser and the upcoming TLS version 1.3 [37], the latter being based on the OPTLS key exchange protocol by Krawczyk and Wee [24]. Those designs set out to establish an initial key in zero round-trip time (0-RTT) that allows one party (usually the client) to send "early" data already along with the first key exchange message to a (previously visited) server.

Without the server being able to contribute, it is well understood that such an approach cannot achieve equally strong security guarantees for the initial key as classical key exchange protocols are able to provide with a full round-trip (and hence contributions from both parties). In particular, the initial key cannot provide (forward) secrecy in a setting where no state is shared between sessions and all but the ephemeral keying material is compromised after the key exchange run. The common strategy is, hence, that both parties switch to a stronger key (e.g., achieving forward secrecy) after the server contributed in a second step of the key exchange and protect any further communication under this key.

DIFFIE–HELLMAN-BASED 0-RTT. One main concept to derive a 0-RTT key based on a Diffie–Hellman-style key exchange and to later upgrade to a stronger, forward-secret key, is shared by both recent prominent instances QUIC and TLS 1.3 (up to `draft-12` [34]).[1] From a high-level perspective (i.e., omitting necessary mechanisms to protect, e.g., against replays or man-in-the-middle attacks which

1. We refer here to the (EC)DHE 0-RTT variant in TLS 1.3 draft `draft-ietf-tls-tls13-12` and the original QUIC proposal `Rev 20130620`, see also our comment in Section 1.3 about the status of these documents.

DOI 10.1109/EuroSP.2017.18

both protocols employ), this concept works as follows. Prior to the actual key exchange, the client is assumed to have talked to the server before and, in that communication, obtained a so-called *server configuration*. Cryptographically speaking, this configuration includes a *semi-static* Diffie–Hellman share g^s, for which the server stores the secret exponent s for a certain time. In QUIC, authentication of this server configuration is via an (offline) signed structure announced by the server, in TLS 1.3 it is signed (online) during a prior handshake.

Within its first message in subsequent executions, the client then sends an ephemeral Diffie–Hellman share g^x, derives the 0-RTT key K_1 as (a function of) $(g^s)^x = g^{xs}$, and is hence immediately able to, e.g., send encrypted data under the key. The server then computes the same key as $(g^x)^s$ (enabling decryption of the 0-RTT data) and responds with its own ephemeral share g^y for the stronger shared key. Both parties derive the full key K_2 as (a function of) g^{xy}, which can then enjoy forward secrecy in the sense that it remains secure even if g^s or the parties long-term secrets are later compromised.

PRESHARED-KEY-BASED 0-RTT. Another concept for establishing a key in zero round-trip time is based on preshared keys (PSKs) and, from `draft-13` [35] on, forms the basis of the only 0-RTT handshake mode specified for TLS 1.3 (i.e., the option for Diffie–Hellman-based 0-RTT was deferred in `draft-13`). Here, the 0-RTT key K_1 is derived from a previously established secret key (e.g., in TLS 1.3 a key established for session resumption in a regular handshake). The client can perform this computation without interaction with the server and hence is able to immediately send encrypted data under K_1. Later, both parties update a full key K_2 derived from the pre-shared secret and further exchanged material, e.g., fresh Diffie–Hellman shares to ensure forward secrecy.

1.2. The Problem with Replays and How It Is (Not) Solved in QUIC and TLS 1.3

The standard approach in key exchange protocols to prevent a man-in-the-middle attacker from replaying messages in order to make a party derive the same key twice is two include a nonce in both the client's and the server's messages and let the nonce contribute to the derived key. For a 0-RTT key exchange, which is essentially a one-pass (i.e., one-message) key exchange protocol [3], messages (and hence keys) are—at first glance—inevitably replayable[2].

The QUIC protocol side-stepped the replay problem in its original cryptographic design [25] (called `Rev 20130620` here) by demanding the server to store all nonces seen in a so-called "strike register"—restricted in size by a server-specific "orbit" prefix and current time contained in the nonces—and rejecting any recurring nonce. As security analyses confirmed [13], [27], this approach indeed allows to establish a secure 0-RTT key which is non-replayable in the sense that no adversary can make a party derive the same key twice. However, while this approach can succeed to prevent *replays on the key-exchange level* (in terms of preventing double-derivation of keys), it inevitably fails to prevent *(logical) replays of the actual data exchanged*, in particular when it comes to real-world settings where a server entity is implemented in a cluster of, potentially distributed, servers, as we explain next. Let us stress that this problem with replays is independent of whether the 0-RTT key exchange is based on Diffie–Hellman or on preshared keys.

As discovered by Daniel Kahn Gillmor in the discussion around the upcoming TLS version 1.3 [31], any 0-RTT anti-replay mechanism deployed at the key exchange level becomes void when combined within an overall channel protocol that aims to provide reliable delivery of data messages (like, e.g., QUIC or TLS). The reason is that such a protocol will resend rejected 0-RTT data under the second (final) key derived automatically in order to ensure delivery. A generic attacker can hence, for any client sending 0-RTT key-exchange messages together with some encrypted data, make this data being delivered twice in the following attack, also illustrated in Figure 1 (see [31] for a more detailed description of the attack).

The attacker first conveys the client's 0-RTT messages and encrypted data to the server (which processes it), but drops the server's key exchange response. It then forces the server to lose its state, e.g., through rebooting, and presents the same messages again to the server. The server, with knowledge about its reset, has to conservatively decline the 0-RTT part of the key exchange for security reasons, but will reply with its own key exchange contribution for the final key which the attacker now simply relays to the client. The client derives the final key and, to ensure reliable delivery, *sends the desired data again* under this key, which the server will hence decrypt and process a second time. This constitutes a replay of the contained application data and might, e.g., result in a web transaction being processed twice.

Note that the contrived requirement that the attacker is able to reboot the server (while the client keeps waiting for a response) vanishes in a real-world scenario with distributed server clusters, where the attacker instead simply forwards the 0-RTT messages to two servers and drops the first server's response. The described attack hence in particular affects the cryptographic design of QUIC, which (among others) specifically targets settings with distributed clusters. Holding up the originally envisioned 0-RTT full replay protection being impossible, Langley and Chang write in the specification of July 2015 [26] (`Rev 20150720`) that this design is "destined to die" and will be replaced by (an adapted version of) the TLS 1.3 handshake. We, however, argue here that QUIC's strategy in `Rev 20130620` still supports some kind of replay resistance, only at a different level. TLS 1.3, in contrast, forgoes any protection mechanisms and instead accepts replays as inevitable (on the channel level). Developers using TLS 1.3 are supposed to

2. We use the notion of replays interchangeably for both messages and the keys computed based on those replayed messages.

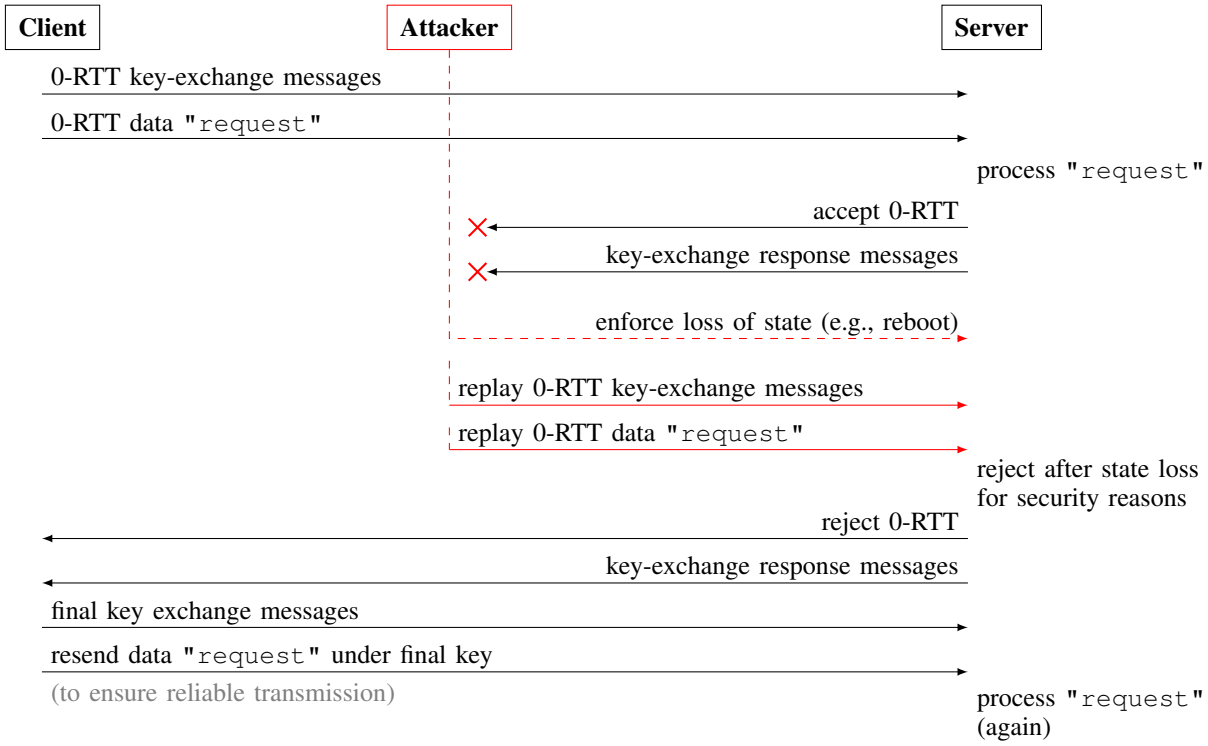


Figure 1. Generic replay attack discovered by Daniel Kahn Gillmor in the IETF TLS working group discussion around TLS 1.3 [31]. The 0-RTT data "request" could, e.g., be an HTTP request "POST /buy-something".

be provided with a different API call for sending 0-RTT data [37, Appendix B.1], indicating its replayability, and responsible for taking replays into account for such data.

There is, then, a significant conceptual gap between replays (of key-exchange messages and keys) on the key-exchange level, and the replay of user data faced on the level of the overall secure channel protocol in the 0-RTT setting. While the former can effectively be prevented within the key exchange protocol, this does not necessarily prevent the latter which can be (and in practice is) induced by the network stack of the channel actively and automatically re-sending (presumably) rejected 0-RTT data under the main key. The latter type of logical, network-stack replays is hence fundamentally beyond of what key exchange protocols can protect against.

1.3. Our Contribution

In this work, we reconsider the derivation of 0-RTT keys in the domain of multi-stage key exchange protocols [13], designed to capture protocols establishing sequences of keys in an intertwined way within a single run. We particularly focus on the security of 0-RTT keys and the question of replays. Within this context, we analyze the preshared-key-based (PSK-based) 0-RTT handshake mode of draft draft-ietf-tls-tls13-14 (short: draft-14) for the upcoming TLS version 1.3 as well as the recently abandoned Diffie–Hellman-based (DH-based) 0-RTT handshake mode in its last specified form in draft-ietf-tls-tls13-12 (short: draft-12).[3] While doing so, we discuss the commonalities and differences between (the original version of) QUIC and the two TLS 1.3 modes in this respect. We stress that, while TLS 1.3 draft-14 already is a relatively mature protocol specification, it remains a draft and still contains un- or underspecified parts. Our inquiry of the TLS 1.3 0-RTT handshakes hence should be conceived as an early (affirmative) discussion of their cryptographic strength, but cannot constitute a definitive analysis.

As mentioned, the Diffie–Hellman-based ((EC)DHE) 0-RTT handshake was removed from the TLS 1.3 draft specification in draft-13, leaving only a PSK-based 0-RTT mode (with or without additional Diffie–Hellman exchange) in the latest drafts. Still, the (EC)DHE 0-RTT variant is much closer to the QUIC and OPTLS proposals, and it may be used as a TLS extension [38], especially since it provides some kind of forward secrecy [21]. We hence also provide an analysis of the DH-based variant to enable a comparison of the security guarantees provided, but focus on the PSK-based 0-RTT handshake specified for draft draft-14.

In more detail, our contributions are fourfold.

COMPARISON OF QUIC AND TLS 1.3. We point out, in passing and explicitly in Section 6, how the designs of QUIC and TLS 1.3 differ in the way of handling 0-RTT,

3. Since our analysis, several follow-up draft versions of TLS 1.3 have been published, maintaining the PSK-based 0-RTT handshake mode.

replay attacks, and data, and how this affects the security. This testifies that, although there may be an agreement on the general goals which should be achieved with 0-RTT, the technical details can vary significantly. One major difference has already been discussed above, carving out that both protocols treat replays differently. Another difference is that QUIC basically restarts the key exchange for an invalid (rejected) 0-RTT request, whereas TLS 1.3 instead only skips over to the regular handshake part. Both protocols also employ different approaches to derive the session keys: While QUIC uses the early key for transmitting both key-exchange messages and application data, TLS 1.3 uses a more versatile approach to create early keys for designated purposes and thus achieves stronger security guarantees and better modularity.

MULTI-STAGE KEY EXCHANGE WITH REPLAYABLE 0-RTT KEYS. As the original QUIC key exchange `Rev 20130620` [25] ensures non-replayability (on the key-exchange level), the analysis by Fischlin and Günther in the multi-stage setting [13] did not consider replays. The TLS 1.3 0-RTT handshake candidates (both `draft-12` DH-based and `draft-14` PSK-based), however, do not aim at preventing replays even on the key-exchange level, motivated by that any such measure would be defeated by the active, logical replay of data ensuring reliable delivery, as discussed above.

We hence extend (in Section 2) the previous multi-stage key exchange models used to analyze QUIC [13] and the full and preshared-key handshakes of TLS 1.3 [10], [12] in several aspects. First of all, we introduce the distinction between replayable and non-replayable stages (and, hence, keys) which, in the case of TLS 1.3, allows us to capture that the 0-RTT key exchange messages of a client session can be replayed to multiple server sessions which will all derive the same key. In order to capture the effects of exposures of the semi-static keys used in a DH-based handshake to non-interactively derive the 0-RTT keys, we allow them to be compromised and define how this affects both 0-RTT keys (which will be compromised) and non–0-RTT keys (which are required to remain secure). We additionally distinguish between keys used to protect application data only (called external keys) and keys which are also used within the key exchange, e.g., to encrypt key exchange messages (called internal keys). Such distinction was previously only made informally for the final key(s) derived [11], [12]. In the TLS 1.3 0-RTT handshakes however, also intermediate keys (namely, the 0-RTT early-data application key tk_{ead}) are only used externally, a setting for which our notion provides a cleaner separation.

SECURITY ANALYSIS OF THE TLS 1.3 DRAFT-14 PSK/PSK-(EC)DHE 0-RTT AND DRAFT-12 (EC)DHE 0-RTT HANDSHAKES. We then apply our model to analyze the PSK and PSK-(EC)DHE 0-RTT handshake modes specified in TLS 1.3 `draft-14` (in Section 4), which we first describe in Section 3, as well as the (EC)DHE 0-RTT handshake mode of `draft-12` (in Section 5). The other specified handshake modes of TLS 1.3, full (EC)DHE and pre-shared key, have already been analyzed by Dowling et al. [12] for the previous (relatively close) `draft-10` [32]. Our analysis shows that all three 0-RTT handshakes are secure (multi-stage) key exchange protocols, establishing random-looking keys. In particular, the two 0-RTT keys derived to protect the early handshake messages and application data, tk_{ehs} resp. tk_{ead}, achieve the desired unilateral resp. mutual authentication, and are—as expected—replayable. Furthermore, we confirm that the second parts of the handshakes (essentially a full (EC)DHE resp. a regular PSK/PSK-(EC)DHE handshake), achieve security similar to that attested by Dowling et al. [12] for `draft-10`. Applying concepts established by Fischlin and Günther [13] and Dowling et al. [10], we show that security holds for the different authentication options of TLS 1.3 running in parallel and that all keys derived are independent in the sense of that leaking one of them does not affect any other key.

Our security results hold under mostly standard cryptographic assumptions like the unforgeability of the signature resp. MAC scheme, collision resistance of the hash function, and pseudorandomness properties of the key derivation function. The handshakes' security further relies on the (plain) pseudorandom-function oracle Diffie–Hellman (PRF-ODH) assumption (introduced and used earlier for the analysis of several Diffie–Hellman–based modes of TLS 1.2 [18], [22]). Notably, for technical reasons that we detail in our proof, we furthermore need to employ a slightly strengthened, double-sided variant msPRF-ODH of the PRF-ODH assumption for the analysis of the (EC)DHE 0-RTT handshake (in `draft-12`).

COMPOSITION WITH EXTERNAL KEYS. The distinction between external(-only) and internal keys finally allows us to establish slightly more general, cleaner composition results (in Section 7). We recall that Fischlin and Günther [13] lifted the Bellare–Rogaway compositional result by Brzuska et al. [5] to the multi-stage setting (later extended by Dowling et al. [10], [12]). On a high level, their result specifies sufficient conditions for a multi-stage key exchange protocol such that the protocol generically composes the established session keys with any symmetric-key protocol. Our refined model determines more clearly which derived session keys can be possibly amenable to this generic composition result, establishing the key being *external* as a necessary condition. We also capture the influence of replays on generic composition, establishing *non-replayability* as a further condition.

1.4. Related Work

Our work builds upon and extends the multi-stage key exchange models by Fischlin and Günther [13] (used to analyze QUIC) and Dowling et al. [10], [11], [12] (used to analyze the full (EC)DHE and preshared-key handshakes of several prior TLS 1.3 drafts).

The practical requirement for reduced round-trip time, nowadays exemplified in the recent designs of QUIC and TLS 1.3, has already appeared in the works about MQV [3]

as well as HMQV [19] and its one-pass version [17]. The idea has later been discussed more formally under the notion of non-interactive key exchange (NIKE) [8], [15]. The difference to 0-RTT key exchange is that NIKEs describe protocols which establish a single session key which is not used within the key exchange. Of course, the key is replayable in the above sense, and security requirements usually disallow trivial attacks on such replayable keys. Another difference to 0-RTT protocols is that for NIKEs there is usually no notion of semi-static keys, i.e., since no further interaction takes place the parties cannot authenticate additional cryptographic keys with limited life span in the subsequent steps.

Krawczyk and Wee [24], [23] recently introduced the OPTLS protocol, which forms the clean and elegant cryptographic core of the TLS 1.3 handshake modes, as well as analyzed its security in the Canetti–Krawczyk model [6], and also proposed its one-pass version [17] for the 0-RTT key. Focusing on the security of the two keys derived in OPTLS (corresponding to the early-data and application traffic keys tk_{ead} and tk_{app} in the TLS 1.3 0-RTT handshakes) separately, the coherence notion of key independence is beyond the scope of their analysis (though mentioned informally), as is the compositional security of these keys. While remarking that a PRF-ODH-like assumption could potentially be used for the proof, Krawczyk and Wee employ different assumptions for their analysis, namely the Gap-DH assumption in the random oracle model. Furthermore, OPTLS does not include a PSK-based 0-RTT mode and TLS 1.3 also extends the OPTLS protocol in order to include client authentication, resumption, encryption of handshake messages, and further exported keying material; aspects that we take into account in our analysis of the 0-RTT handshake modes.

A recent work by Hale et al. [16] introduces a simplified security model for low-latency key exchange with two session keys, one early key and one final key, as in case of QUIC. For their security model they introduce a notion called strong key independence which basically says that revealing the early key does not violate secrecy of the final key (or vice versa). This seems to be exactly in the same spirit as the notion of key independence introduced earlier in [13] and also used here. Furthermore, Hale et al. [16] point out that QUIC does not provide strong key independence. This has already been discussed in [13], inciting the authors in [13] to also propose a slight modification of QUIC, called QUIC*i*, which achieves their notion of key independence via modifying the key derivation steps only. Hale et al. [16] then give a new generic construction secure in their model, including strong key independence, based on non-interactive key exchange. Finally, as discussed in [16], the back then available `draft-08` still had a not fully specified 0-RTT mode, leaving the implications of the results to the current draft of TLS 1.3 unclear. Remarkably, the model allows for replay attacks on the 0-RTT keys, as a consequence confining the admissible tests on keys, even though such replay attacks are excluded in QUIC via strike registers. Moreover, the latest drafts of TLS 1.3 generate more than two session keys, as considered in the model of [16].

Also recently, Cremers et al. [9] presented a tool-supported analysis of TLS 1.3 `draft-10` including 0-RTT mode, resumption, and delayed client authentication, discovering an attack on the interaction between the preshared-key handshake and the (expected) specification of client authentication.

Finally, our work is part of a substantial effort of the security research community in analyzing draft versions of TLS 1.3 prior to its standardization. We refer to Paterson and van der Merwe [28] for an overview over these analyses and a discussion of TLS 1.3's proactive standardization process.

2. Modeling Replayable 0-RTT in Multi-Stage Key Exchange

In this section we recap the *multi-stage key exchange* model introduced by Fischlin and Günther [13], and later extended by Dowling et al. [10], [12], and augment it for capturing replayable 0-RTT keys. To capture security of the `draft-12` (EC)DHE 0-RTT handshake (where authentication is provided through long-term signing keys) as well as the `draft-14` PSK-based 0-RTT handshakes, we treat both the public-key (MSKE) and the preshared-secret (MS-PSKE) variant of the model. For space reasons, we can only outline the original multi-stage setting and focus on the modifications we introduce; we refer to [13], [10], [12] for the formal definition of the original multi-stage security models.

2.1. Outline

The multi-stage key exchange model follows the game-based paradigm of Bellare and Rogaway [2]. That is, the adversary controls the network over which the parties communicate, giving the adversary the power to read and alter protocol messages in transmission. For this the adversary can call a NewSession oracle (starting a new session of a specified honest party) and a Send oracle (which delivers some message to a specified party). To argue about the secrecy of keys the adversary may make (multiple) Test queries for some stage of the protocol to either receive the corresponding session key of that stage or to obtain an independent random key instead. In the multi-stage setting one must restrict the set of admissible Test queries to avoid trivial attacks, e.g., if a tested session key is used in a later stage of the execution.

To model leakage of long-term secrets and, through this, forward secrecy of keys (i.e., security after long-term secret compromise) we also grant the adversary access to a Corrupt oracle which returns the corresponding secret key. Moreover, we (independently) capture leakage of semi-static keys (used in Diffie–Hellman-based 0-RTT key derivation as, e.g., TLS 1.3 `draft-12` (EC)DHE 0-RTT) through a separate RevealSemiStaticKey oracle. In our model we do not consider leakage of internal state of the parties (such as randomness or master secrets) but note that one can in principle enhance the model further to capture such attacks. Still,

we allow for leakage of session keys, modeling insecure usage of such keys in follow-up protocol steps (of the key exchange protocol itself or in subsequent communication protocols). As in the common Bellare–Rogaway model we must prohibit compromise of secrets in sessions partnered with tested sessions. Here, partners are identified via session identifiers.

Session identifiers in the multi-stage setting are more elaborate than in the single-stage Bellare–Rogaway case. Depending on the protocol, revealing a session key via a Reveal query may render the subsequent session key insecure, e.g., if contributions to the session key of the following stage are sent authenticated under the key of the current stage (as in QUIC [26], [13]). A protocol which can tolerate such leakage is called (session-)key independent, else it is called key dependent.

For TLS 1.3, Dowling et al. [10] refined the original model [13] (beyond introducing the preshared-secret variant) further to cover the various authentication properties of the different handshakes and also of the different stages. For this they distinguish between unauthenticated, unilaterally authenticated, and mutually authenticated stages, and treat security of multiple sessions running in parallel with different authentication modes. They also implemented the common TLS property of post-specified peers [7] via wildcards '$*$' for intended communication partners, which can be set once throughout the protocol run, e.g., after the party has verified the certificate of the partner.

Another change from [13] to [10] concerns the notion of contributive identifiers for the case of sessions with unauthenticated partners. Since the adversary can potentially impersonate the unauthenticated partner, keys in such sessions cannot be secure in general. Still, such keys should be considered secure as long as the full contribution to the key clearly stems from an honest party (even though this honest party may never complete its execution to output a session identifier). Contributive identifiers allow to specify such partnered contributions.

Both works [13], [10] follow the approach of Brzuska et al. [5] to split the security requirements into one for Match security, capturing among others uniqueness of session identifiers, correspondence of session identifiers and keys, and linking contributive identifiers to session identifiers, and into the common requirement for key secrecy.

2.2. Adding 0-RTT to Multi-Stage Protocols

To capture 0-RTT in the multi-stage setting we augment the model in [10], [12] by the following points:

1) We introduce the notion of replayable stages.
2) We allow exposure of the semi-static keys used for establishing 0-RTT keys. Note that the exposure of pre-shared keys (used for PSK-based 0-RTT) is already captured through Corrupt queries.
3) We distinguish between external session keys (used for protecting the application layer only) and internal session keys (which can be used within the key exchange protocol). Note that since, by default, session keys are always output by key exchange protocols, internal keys may thus also be used further in applications, of course.

As mentioned in the introduction, replayable stages in a multi-stage key exchange protocol are basically those stages in which an adversary can force more than two sessions to share the same session identifiers and session keys by replaying previous interactions. Note that for 0-RTT, replayability is inevitable for stateless parties, whereas Google's QUIC protocol in version `Rev 20130620` thwarts such replay attacks via strike registers storing information about previous connections.

In our model we assume that the protocol specifies stages as replayable or non-replayable. The latter type of stage leaves the original security properties untouched. The replayable kind of stage allows for multiple collisions among session identifiers and keys, such that we need to relax the notion of Match security for such stages. Key secrecy should be not affected by replayability because the adversary may be able to foist the same key on multiple sessions, but the key itself should still look random.

The other modification refers to exposure of semi-static keys. Recall that such keys are used in TLS 1.3 (up to `draft-12`) to enable 0-RTT for the DH-based mode by having the client mix a fresh ephemeral key to such a semi-static key. The life span of such a semi-static key may range over multiple sessions, and we therefore leave the choice of which of these keys to use (and when to create it) to the adversary. This is modeled by augmenting the NewSession query by a field for specifying the semi-static key, and by having a NewSemiStaticKey command to let a party create a fresh semi-static key. Leakage of semi-static keys is captured by adding a RevealSemiStaticKey query to the model through which the adversary learns the corresponding secret key. The previous works [13], [10] have not yet supported such leakage, even though [13] already introduced the equivalent idea of a temporary key for analyzing QUIC.

Note that we separate leakage of the semi-static key by RevealSemiStaticKey queries from revealing the long-term secret via Corrupt queries. This corresponds to the setting where a semi-static key may actually no longer be used by a server and its secret key been irrevocably erased. The adversary can, of course, always mount RevealSemiStaticKey commands before a Corrupt request, thus enhancing the adversary's capabilities and strengthening the security claims.

The explicit distinction between internal and external session keys implements a cleaner way to deal with early derived keys used exclusively in, e.g., the TLS record protocol. In contrast to QUIC, where only the final session key is not used in the key exchange messages, the 0-RTT step of TLS 1.3 allows both parties to immediately establish the early handshake and early application-data traffic key tk_{ehs} and tk_{ead}, with a clear separation of concerns. The former early handshake key is used to protect the handshake messages (internally), and the early application-data key is only used to protect external application data sent by the client before finishing the full handshake.

In contrast to previous works formalizing multi-stage key exchange [13], [10], [12], we explicitly separate some

protocol-specific properties (as, e.g., various authentication flavors) from *session-specific* properties (as, e.g., the state of a running session). We represent protocol-specific properties as a vector $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ that captures the following:

- $\mathsf{M} \in \mathbb{N}$: the number of stages (i.e., the number of keys derived)[4]
- $\mathsf{AUTH} \subseteq \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}^{\mathsf{M}}$: the set of supported authentication properties (for each stage). As in [10] we call stages and keys *unauthenticated* if they provide no authentication for either communication partner, *unilaterally authenticated* if they authenticate only the responder (server) side, and *mutually authenticated* if they authenticate both communication partners.
- $\mathsf{USE} \in \{\mathsf{internal}, \mathsf{external}\}^{\mathsf{M}}$: the usage indicator for each stage, where USE_i indicates the usage of the stage-i key. Here, an internal key is used within the key exchange protocol (but possibly also externally), whereas an external key must not be used within the protocol, making the latter potentially amenable to generic composition (see Section 7).
- $\mathsf{REPLAY} \in \{\mathsf{replayable}, \mathsf{nonreplayable}\}^{\mathsf{M}}$: the replayability indicator for each stage, where REPLAY_i indicates whether the i-th stage is replayable in the sense that a party can easily force identical communication and thus identical session identifiers and keys in this stage (e.g., re-sending the same data in 0-RTT stages). Note that the adversary, however, should still not able to distinguish such a replayed key from a random one. Note that, from a security viewpoint, the usage of replayable stages should ideally be small, whereas such stages usually come with an efficiency benefit.

2.3. Security of Multi-Stage Key Exchange

The security properties for multi-stage key exchange protocols are almost identical with those given for the TLS 1.3 full and resumption handshake analysis by Dowling et al. [10]; split into two games following Fischlin and Günther [13] and Brzuska et al. [5], [4]. On the one hand, Match security ensures that the session identifiers sid effectively match the partnered sessions. On the other hand, Multi-Stage security ensures Bellare–Rogaway-like key secrecy.

For the analysis of the TLS 1.3 0-RTT handshakes, most changes in the model are already reflected in the specification of the adversarial queries. Beyond introducing the new protocol-specific properties and the RevealSemiStaticKey query in the definition, we only extend the Match security definition in order to allow for multiple sessions being partnered in replayable stages. Due to space limitations, we only describe briefly the adapted Match and Multi-Stage security notions and refer to the full version of this work [14] for the formal definitions.

2.3.1. Match **Security.** The notion of Match security—which we adapt from [10], [12] to capture replayable stages—ensures soundness of the session identifiers sid, i.e., that they properly identify the partnered sessions in the sense that they share the same session keys, authentication, and contributive identifier as well as agree on the intended (authenticated) partner and pre-shared secret (if used). Furthermore, session identifiers must not coincide across different stages, and for non-replayable stages session identifiers must not appear in more than two sessions. For replayable stages the latter requirement is dropped.

2.3.2. Multi-Stage **Security.** The second notion, Multi-Stage security, captures Bellare–Rogaway-like key secrecy in the multi-stage setting. Our modification in particular gives the adversary access to the NewSemiStaticKey and RevealSemiStaticKey queries and states security wrt. the introduced protocol-specific properties $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$.

3. The TLS 1.3 draft-14 PSK and PSK-(EC)DHE 0-RTT Handshake Protocols

Starting from `draft-13`, TLS 1.3 only specifies PSK-based 0-RTT handshake modes, abandoning the (EC)DHE-based variant predominant in earlier drafts. We hence first analyze the preshared-key–based variants, PSK(-only) 0-RTT and PSK-(EC)DHE 0-RTT, as specified in TLS 1.3 `draft-14` [36]. In the PSK 0-RTT mode, keys are solely derived from a beforehand established pre-shared key (usually the resumption master secret RMS derived in a full TLS 1.3 handshake). In the PSK-(EC)DHE 0-RTT mode, (elliptic curve) Diffie–Hellman shares additionally enter the key derivation. Our analysis of the purely Diffie–Hellman-based (EC)DHE 0-RTT mode (of `draft-12`) is stated later in Section 5.

The TLS 1.3 0-RTT handshake protocols can be conceptually subdivided into four phases:

Key exchange. In the key exchange phase, parties negotiate the ciphersuites and key-exchange parameters to be used and establish shared key material as well as traffic keys to encrypt the remaining handshake.

0-RTT. In the 0-RTT (data) phase, which is interleaved with the key exchange phase, the client can send application data already in its first flight. For this purpose, traffic keys for encrypting the early handshake and application data are established.[5]

Server parameters. In the server parameters phase, further handshake parameters (as, e.g., whether client authentication is demanded) are fixed by the server.

Authentication. In the authentication phase, both the server and client can (based on the aspired authentication)

4. We fix a maximum stage M only for ease of notation. Note that M can be arbitrarily large in order to cover protocols where the number of stages is not bounded a-priori. Also note that, for technical convenience, stages and session keys may be "back to back," without further protocol interactions between parties.

5. For comparison, omitting the 0-RTT phase essentially yields the TLS 1.3 full resp. PSK-based handshake.

authenticate, verify that they share the same view of the handshake, and derive (authenticated) application traffic keys.

We illustrate the protocol flow (with the cryptographically relevant computations) and the key schedule for both the PSK(-only) and PSK-(EC)DHE 0-RTT handshakes in Figure 2. The handshake messages are the following:

- `ClientHello` (`CH`)/`ServerHello` (`SH`) contains random nonces r_c / r_s of the respective parties, as well as negotiation parameters (supported versions and ciphersuites). Several extensions can be sent along with the `Hello` messages; the PSK and PSK-(EC)DHE 0-RTT handshakes require the following two resp. three extensions to be included.
- `ClientEarlyData` (`CEAD`)/`ServerEarlyData` (`SEAD`) are extensions sent to announce a 0-RTT handshake. The client includes the (masked) age `ticket_age` of the (ticket issuing the) used resumption secret. The server signals accepting the 0-RTT exchange with an empty `ServerEarlyData` extension.
  TLS 1.3 `draft-14` [36, Section 4.2.6.2] recommends that servers should use the `ticket_age` value to check that client messages are not replayed. Depending on how well clocks are synchronized, this can prevent delayed replays, but not immediate replays. We do not rely on this check in our analysis but conservatively treat the 0-RTT key exchange messages as arbitrarily replayable.
- `ClientPreSharedKey` (`CPSK`)/`ServerPreSharedKey` (`SPSK`) are extensions in which the client announces one (or multiple) pre-shared key identifier(s) (`psk_id`), of which the server selects one to be used as the pre-shared secret (pss) in the handshake. Focusing on 0-RTT handshakes only, we only consider the case where client and server agree on the first announced `psk_id`, the one used to derive 0-RTT keys.
- `ClientKeyShare` (`CKS`)/`ServerKeyShare` (`SKS`) are extensions sent only for the PSK-(EC)DHE 0-RTT handshake. They contain the ephemeral Diffie–Hellman shares $X = g^x$ resp. $Y = g^y$ for several (in case of the client) or one group (the server decided for).

Based on these messages both sides can already derive the 0-RTT keys. First, from the shared pre-shared secret (the resumption master secret established in a previous handshake) $\mathsf{pss} = \mathrm{RMS}$ a pre-shared key PSK and a resumption context value rctxt are derived using the Expand component of HKDF [20]. In the key derivation, PSK serves as the starting secret while rctxt binds the derived keys to the previous handshake that established RMS. Then, the early secret ES is computed from PSK using HKDF.Extract. Via an intermediate (expanded) early traffic secret ETS both the 0-RTT handshake and application traffic keys tk_{ehs} and tk_{ead} are finally expanded.

For the two HKDF functions we use the following common notation: Function $\mathsf{HKDF.Extract}(\mathit{XTS}, \mathit{SKM})$ obtains as input a (not necessarily secret and potentially fixed) extractor salt *XTS* and some source key material *SKM*, and outputs a pseudorandom key PRK. Function $\mathsf{HKDF.Expand}(\mathit{PRK}, \mathit{CTXinfo})$ obtains as input a pseudorandom key *PRK* (here: the output of the Extract step) and some (potentially empty) context information *CTXinfo*, and outputs some key material *KM*.[6] Both functions are based on HMAC [1].

The client then completes its first flight by sending a 0-RTT `Finished` message, sent encrypted under tk_{ehs}:

- $\mathtt{ClientFinished}_0$ ($\mathtt{CF}_0$) consists of an HMAC (message authentication code) value which is computed using the 0-RTT finished secret $\mathrm{FS}_{\text{0-RTT}}$ on the (hashed) 0-RTT messages and the resumption context rctxt.

Following $\mathtt{ClientFinished}_0$, the client can use tk_{ead} to encrypt and send 0-RTT application data.[7]

After receiving the client's first flight, the server sends its `ServerHello` message along with the indicated extensions. At this point (resp. after receiving `ServerHello` for the client) both sides extract from ES the handshake secret HS (incorporating the joint Diffie–Hellman share $\mathrm{DHE} = g^{xy}$ in the PSK-(EC)DHE 0-RTT handshake). Again via first expanding an intermediate handshake traffic secret HTS, the handshake traffic key tk_{hs} is derived.

Server and client then complete the handshake by sending the following messages encrypted under tk_{hs}:

- `EncryptedExtensions` (`EE`), sent by the server, allows to specify further extensions.
- `ClientFinished` (`CF`)/`ServerFinished` (`SF`) contain an HMAC value over the handshake hash, keyed with the client resp. server finished secret $\mathrm{FS_C}$/$\mathrm{FS_S}$ which are both expanded from the handshake traffic secret HTS.

At the end of the handshake, the master secret MS is extracted from HS and used to expand the application traffic key tk_{app} (via an intermediate traffic secret TS), used to protect the (non–0-RTT) application data sent, as well as the exporter master secret EMS which can be used to derive further key material outside of TLS.

ON CLIENT AUTHENTICATION, 0.5-RTT DATA, AND POST-HANDSHAKE MESSAGES. When analyzing the (PSK-based) 0-RTT handshake candidates for TLS 1.3, we focus on the main components of the handshake and hence do not capture the following more advanced options specified in `draft-14`.

First, the server can optionally ask the *client to authenticate* (beyond the shared secret key) by sending a public-key certificate and signing the transcript (i.e., by signature-based authentication as employed in the (EC)DHE-based handshakes of TLS 1.3).[8] We omit this option in our analysis

6. We assume the third, output-length parameter L in the Expand function to be fixed to $L = \lambda$ for our security parameter λ and hence always omit it.

7. The server may decide to not derive any 0-RTT keys (and not accept any 0-RTT data). In that case it would, in our model, simply set the first two session identifiers sid_1, sid_2 and keys K_1, K_2 to $\perp$ and continue with deriving the third key.

8. There is also discussion to further include signature-based server authentication in the PSK-based 0-RTT handshakes [33].

Client | **Server**

$\mathsf{PSK} \leftarrow \mathsf{HKDF.Expand}(\mathrm{RMS}, \mathrm{label}_1)$
$\mathrm{rctxt} \leftarrow \mathsf{HKDF.Expand}(\mathrm{RMS}, \mathrm{label}_2)$

$\mathtt{ClientHello}$: $r_c \xleftarrow{\$} \{0,1\}^{256}$
$+\ \mathtt{ClientEarlyData}$: $\mathtt{ticket\_age}$
$+\ \mathtt{ClientPreSharedKey}$: $\mathtt{psk\_id}_1, \ldots$
$[+\ \mathtt{ClientKeyShare}$: $X \leftarrow g^x]^\dagger$

$\longrightarrow$

$\mathrm{ES} \leftarrow \mathsf{HKDF.Extract}(0, \mathrm{PSK})$
$H_1 \leftarrow \mathsf{H}(\mathtt{CH}) || \mathsf{H}(\mathrm{rctxt})$
$\mathrm{ETS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ES}, \mathrm{label}_3 || H_1)$
$tk_{ehs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_4)$ stage 1

$(\mathtt{ClientFinished}_0)$:
$\mathrm{FS}_{0\text{-RTT}} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_5)$
$\mathtt{CF}_0 \leftarrow \mathsf{HMAC}(\mathrm{FS}_{0\text{-RTT}}, H_1)$

$\longrightarrow$

check $\mathtt{CF}_0 = \mathsf{HMAC}(\mathrm{FS}_{0\text{-RTT}}, H_1)$
$tk_{ead} \leftarrow \mathsf{HKDF.Expand}(\mathrm{ETS}, \mathrm{label}_6)$ stage 2

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

$\mathtt{ServerHello}$: $r_s \xleftarrow{\$} \{0,1\}^{256}$
$+\ \mathtt{ServerEarlyData}$
$+\ \mathtt{ServerPreSharedKey}$: $\mathtt{psk\_id}$
$[+\ \mathtt{ServerKeyShare}$: $Y \leftarrow g^y]^\dagger$

$\longleftarrow$

$H_2 \leftarrow \mathsf{H}(\mathtt{CH}||\mathtt{SH}) || \mathsf{H}(\mathrm{rctxt})$
$[\mathrm{DHE} \leftarrow Y^x]^\dagger$ $[\mathrm{DHE} \leftarrow 0]^\diamond$ $[\mathrm{DHE} \leftarrow X^y]^\dagger$
$\mathrm{HS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{ES}, \mathrm{DHE})$
$\mathrm{HTS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HS}, \mathrm{label}_7 || H_2)$
$tk_{hs} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_8)$ stage 3

$\{\mathtt{EncryptedExtensions}\}$
$H_3 \leftarrow \mathsf{H}(\mathtt{CH}||\mathtt{SH}||\mathtt{EE}) || \mathsf{H}(\mathrm{rctxt})$
$\mathrm{FS_S} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_9)$
$\{\mathtt{ServerFinished}\}$:
$\mathtt{SF} \leftarrow \mathsf{HMAC}(\mathrm{FS_S}, H_3)$

$\longleftarrow$

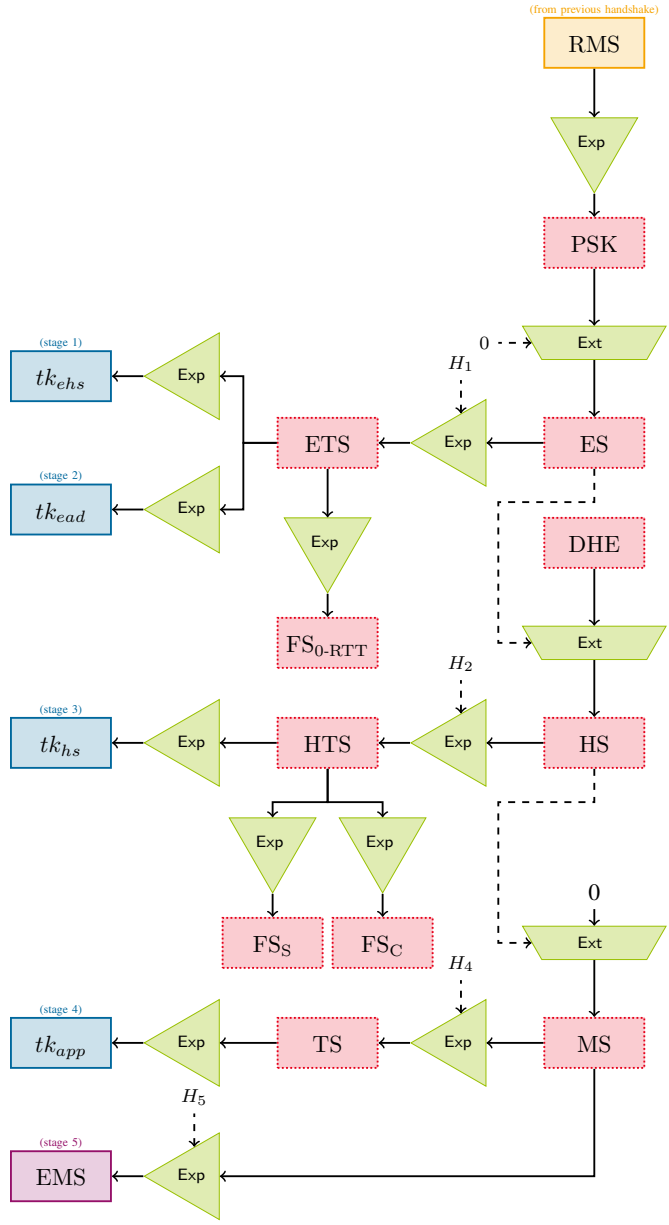
check $\mathtt{SF} = \mathsf{HMAC}(\mathrm{FS_S}, H_3)$
$H_4 \leftarrow \mathsf{H}(\mathtt{CH}||\mathtt{SH}||\mathtt{EE}||\mathtt{SF}) || \mathsf{H}(\mathrm{rctxt})$
$\mathrm{FS_C} \leftarrow \mathsf{HKDF.Expand}(\mathrm{HTS}, \mathrm{label}_5)$
$\{\mathtt{ClientFinished}\}$:
$\mathtt{CF} \leftarrow \mathsf{HMAC}(\mathrm{FS_C}, H_4)$

$\longrightarrow$

check $\mathtt{CF} = \mathsf{HMAC}(\mathrm{FS_C}, H_4)$
$H_5 \leftarrow \mathsf{H}(\mathtt{CH}||\mathtt{SH}||\mathtt{EE}||\mathtt{SF}||\mathtt{CF}) || \mathsf{H}(\mathrm{rctxt})$
$\mathrm{MS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{HS}, 0)$
$\mathrm{TS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_{10} || H_4)$

$tk_{app} \leftarrow \mathsf{HKDF.Expand}(\mathrm{TS}, \mathrm{label}_{11})$ stage 4
$\mathrm{EMS} \leftarrow \mathsf{HKDF.Expand}(\mathrm{MS}, \mathrm{label}_{12} || H_5)$ stage 5

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



Protocol flow legend

$\mathtt{MSG}$: Y	TLS 1.3 message $\mathtt{MSG}$ containing Y
$(\mathtt{MSG})$	message $\mathtt{MSG}$ AEAD-encrypted with tk_{ehs}
$\{\mathtt{MSG}\}$	message $\mathtt{MSG}$ AEAD-encrypted with tk_{hs}
$+\ \mathtt{MSG}$	message sent as extension within previous message
$[\ldots]^\dagger$	message/computation only in PSK-(EC)DHE 0-RTT
$[\ldots]^\diamond$	message/computation only in PSK 0-RTT

Key schedule legend

Ext / Exp	HKDF.Extract resp. HKDF.Expand function
X - - →	salt input X to Ext resp. context info input X to Exp (label inputs to Exp are omitted)

Figure 2. The TLS 1.3 `draft-14` PSK and PSK-(EC)DHE 0-RTT handshake protocols (left) and key schedule (right).

but note that our multi-stage key exchange model can in principle be augmented to capture combined authentication under multiple long-term secrets.

Second, instead of deriving the application traffic key tk_{app} at the end of the handshake (as depicted in Figure 2), the server might already do so after sending the `ServerFinished` message in order to send so-called *0.5-RTT data* directly following his flight, i.e., without waiting for the `ClientFinished` response. We omit analyzing this variant of the handshake but expect that results for it with potentially weaker authentication guarantees for tk_{app} can be obtained in our model.

Third, TLS 1.3 introduces *post-handshake messages* that can be sent (potentially long) after the initial handshake was completed in order to update the used traffic key, authenticate the client, or issue tickets for session resumption. Here, we focus on the main handshake and do not consider post-handshake messages.

4. Security of the TLS 1.3 draft-14 PSK and PSK-(EC)DHE 0-RTT Handshakes

Our security analysis of the TLS 1.3 `draft-14` PSK and PSK-(EC)DHE 0-RTT handshakes (`draft-14-PSK-0RTT` resp. `draft-14-PSK-DHE-0RTT`) is carried out in the preshared-secret variant (MS-PSKE) of the multi-stage key exchange model. We begin with stating the protocol-specific properties $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ mostly shared by both handshakes:

- $\mathsf{M} = 5$: the PSK-based 0-RTT handshakes have five stages (deriving, in that order, keys tk_{ehs}, tk_{ead}, tk_{hs}, tk_{app}, and EMS).
- the authentication properties AUTH differ between the PSK(-only) and the PSK-(EC)DHE 0-RTT handshakes:
  - for PSK 0-RTT, $\mathsf{AUTH} = \{(\mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual}, \mathsf{mutual})\}$: all keys established are mutually authenticated (wrt. the established pre-shared secret).
  - for PSK-(EC)DHE 0-RTT, $\mathsf{AUTH} = \{(\mathsf{mutual}, \mathsf{mutual}, \mathsf{unauth}, \mathsf{mutual}, \mathsf{mutual})\}$: the handshake traffic key tk_{hs} is unauthenticated, all other keys are mutually authenticated (wrt. the established pre-shared secret).[9]
- $\mathsf{USE} = (\mathsf{internal}, \mathsf{external}, \mathsf{internal}, \mathsf{external}, \mathsf{external})$: the (0-RTT and main) handshake traffic keys tk_{ehs} and tk_{hs} are used to protect messages within the handshake while the application traffic keys tk_{ead} and tk_{app} as well as the exporter master secret EMS are only used externally.
- $\mathsf{REPLAY} = (\mathsf{replayable}, \mathsf{replayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable}, \mathsf{nonreplayable})$: the 0-RTT stages 1 and 2 are replayable, the other stages are not.

Both TLS 1.3 `draft-14` PSK-based 0-RTT handshakes enjoy key independence for all keys. Expectedly, the PSK(-only) 0-RTT handshake provides no forward secrecy. The PSK-(EC)DHE 0-RTT handshake instead ensures forward secrecy for the non–0-RTT keys (i.e., from stage 3 on), but not for the 0-RTT keys.

Session matching is defined via the following session identifiers, consisting of the unencrypted messages exchanged up to each stage: $\mathsf{sid}_1 = (\mathtt{CH})$, $\mathsf{sid}_2 = (\mathsf{sid}_1, \text{“EAD”})$, $\mathsf{sid}_3 = (\mathtt{CH}, \mathtt{SH})$, $\mathsf{sid}_4 = (\mathtt{CH}, \mathtt{SH}, \mathtt{EE}, \mathtt{SF})$, and $\mathsf{sid}_5 = (\mathtt{CH}, \mathtt{SH}, \mathtt{EE}, \mathtt{SF}, \mathtt{CF})$. Here, `Hello` messages also comprise the sent `EarlyData`, `KeyShare`, and `PreSharedKey` extensions. We remark that, as for the analysis of the TLS 1.3 full and resumption handshake [10], we too define the session identifiers over the *unencrypted* messages. This diverges from the common practice to set the session identifier as the concatenation of the (here encrypted) protocol transmissions, but is necessary to achieve key independence in the multi-stage security for such protocols.

For the contributive identifiers, we need to ensure that a server session can in any case be tested when receiving an honest client contribution (even if that client never receives the `ServerHello` response), analogously to the full handshake analysis in [10]. Hence, for stage 3, on sending resp. receiving the `ClientHello` message, client resp. server initially sets $\mathsf{cid}_3 = (\mathtt{ClientHello})$ and subsequently, on receiving (resp. sending) the `ServerHello` message, extend it to $\mathsf{cid}_3 = (\mathtt{ClientHello}, \mathtt{ServerHello})$. All other contributive identifiers are set to $\mathsf{cid}_i = \mathsf{sid}_i$ when the respective stage's session identifier is set.

We are now ready to state our security results for the PSK and PSK-DHE 0-RTT handshakes of TLS 1.3 `draft-14`. Naturally, the proof aspects concerning the non–0-RTT parts of the handshakes are structurally close to the proofs for the `draft-10` PSK-based handshakes by Dowling et al. [12], but need to take the modified key schedule into account. Due to space limitations, we only present abridged versions of our proofs here and refer to the full version [14] for the complete proofs.

4.1. PSK(-only) 0-RTT Handshake

Theorem 4.1 (Match security of `draft-14-PSK-0RTT`)**.** *The* `draft-14` *PSK 0-RTT handshake is* Match*-secure: for any efficient adversary $\mathcal{A}$ we have* $\mathsf{Adv}^{\mathsf{Match}}_{\mathtt{draft\text{-}14\text{-}PSK\text{-}0RTT},\mathcal{A}} \leq n_s^2 \cdot 2^{-|nonce|}$*, where n_s is the maximum number of sessions and $|nonce| = 256$ is the bit-length of the nonces.*

Proof (sketch). For space reasons, we omit the proof details here and just note that the bound $n_s^2 \cdot 2^{-|nonce|}$ capture the probability of three honest sessions colliding on a session identifier due to same nonces. □

9. Although including the pre-shared secret in the derivation of tk_{hs}, as the involved Diffie–Hellman shares are only authenticated after its derivation, tk_{hs} cannot enjoy both forward secrecy and mutual authentication. We remark that alternatively to considering tk_{hs} being unauthenticated but forward-secret (a security property close to the notion of “weak (perfect) forward secrecy” [19]), one might instead also consider tk_{hs} to be non–forward-secret but mutually authenticated.

Theorem 4.2 (Multi-Stage security of `draft-14-PSK-0RTT`)**.** *The* `draft-14` *PSK 0-RTT handshake is* Multi-Stage*-secure in a key-independent and non–forward-secret manner with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. Formally, for any efficient adversary* $\mathcal{A}$ *against the* Multi-Stage *security there exist efficient algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_9$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-14-PSK-0RTT},\mathcal{A}} \leq 5n_s \cdot \Big(&\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} \\ + n_p \cdot \Big(&\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4} \\ &+ \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_7} \\ &+ \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_9}\Big)\Big),\end{aligned}$$

where n_s *is the maximum number of sessions and* n_p *is the maximum number of pre-shared secrets.*

Proof (abridged). We first restrict the adversary $\mathcal{A}$ to a single Test query (known in advance), reducing its advantage by a hybrid argument (cf. Dowling et al. [11, Appendix A]) by at most $1/5n_s$. Our proof then proceeds via the following sequence of games.

Game 1. We first exclude hash collisions in the execution of honest sessions. By having an algorithm $\mathcal{B}_1$ act as the challenger in the original game and output, when they occur, the two colliding inputs as a collision for H, we can bound the probability of aborting by $\mathcal{B}_1$'s advantage in breaking the collision resistance of H, denoted by $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$.

Game 2. Next, we guess the pre-shared secret $\mathsf{pss} = \mathrm{RMS}$ employed in the tested session, reducing the advantage of $\mathcal{A}$ by a factor of at most the number of pre-shared secrets n_p.

We can now, one at a time, replace the outputs of HKDF.Expand and HKDF.Extract evaluations using RMS and derived keys by random values, leading to a sequence of according advantage bounds for their PRF security or randomness bounds of the underlying HMAC function.

Game 3. We begin by replacing any HKDF.Expand application using $\mathsf{pss} = \mathrm{RMS}$ by evaluations of a (lazy-sampled) random function, which in particular leads to PSK and rctxt being replaced by random values $\widetilde{\mathrm{PSK}}, \widetilde{\mathrm{rctxt}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested (and any partnered) session.

The introduced difference in the advantage of $\mathcal{A}$ can be bounded by an adversary $\mathcal{B}_2$ against the PRF security of HKDF.Expand as follows. Algorithm $\mathcal{B}_2$ simulates Game 2 faithfully, but uses its PRF oracle for evaluating HKDF.Expand under RMS. Note that all keys derived in the PSK 0-RTT handshake are non–forward-secret and hence any (successful) adversary $\mathcal{A}$ cannot issue a Corrupt query on $\mathsf{pss} = \mathrm{RMS}$ used in the tested session. As $\mathcal{B}_2$ hence (perfectly) simulates either Game 2 or Game 3, this step can be bounded by $\mathcal{B}_2$'s advantage $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_2}$.

Game 4. Next, we replace values ES computed as $\mathsf{HKDF.Extract}(0, \widetilde{\mathrm{PSK}})$ by a random value $\widetilde{\mathrm{ES}} \xleftarrow{\$} \{0,1\}^\lambda$, in particular in the tested session and partnered sessions.

Recall that $\mathsf{HKDF.Extract}(\mathit{XTS}, \mathit{SKM})$ is defined as $\mathsf{HMAC}(\mathit{XTS}, \mathit{SKM})$ [20]. Assuming that for any polynomial-time algorithm $\mathcal{B}_3$ it is computationally hard to distinguish $\mathsf{HMAC}(0, \mathit{SKM})$ from $X \xleftarrow{\$} \{0,1\}^\lambda$ for uniformly random chosen values $\mathit{SKM} \in \{0,1\}^\lambda$, we can bound this by the distinguishing advantage of $\mathcal{B}_3$ (acting as the challenger for $\mathcal{A}$), which we denote by $\mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3}$.

Game 5. We next replace evaluations of HKDF.Expand keyed with $\widetilde{\mathrm{ES}}$ as well as HKDF.Extract using $\widetilde{\mathrm{ES}}$ as salt by random functions. This in particular replaces, in the tested and partnered sessions, the early traffic and handshake secret in these sessions by random values $\widetilde{\mathrm{ETS}}, \widetilde{\mathrm{HS}} \xleftarrow{\$} \{0,1\}^\lambda$.

Observe that in both replaced Extract and Expand evaluations, $\widetilde{\mathrm{ES}}$ is (by definition of HKDF) used to key the HMAC function, applied to H_1 (when expanding ETS) resp. to a fixed value 0 (when extracting HS), i.e., distinct inputs. We can hence bound this step by the PRF security of HMAC, denoted by $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4}$.

Game 6. As the next step, we replace HKDF.Expand evaluations keyed with $\widetilde{\mathrm{ETS}}$ (unique to the tested and stage-1-partnered sessions, as derived (without hash collisions) from H_1) by a lazy-sample random function, resulting in random values $\widetilde{tk_{ehs}}, \widetilde{tk_{ead}}, \widetilde{\mathrm{FS}_{\text{0-RTT}}} \xleftarrow{\$} \{0,1\}^\lambda$. This step can similarly to Game 3 be bounded by $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5}$.

Game 7. We again in parallel replace both HKDF.Expand and HKDF.Extract evaluations, this time keyed resp. salted with $\widetilde{\mathrm{HS}}$ by random function, leading to random values $\widetilde{\mathrm{HTS}}, \widetilde{\mathrm{MS}} \xleftarrow{\$} \{0,1\}^\lambda$. As for Game 5, this step is bounded by $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_6}$.

Game 8. We now replace evaluations of HKDF.Expand using $\widetilde{\mathrm{HTS}}$ (again unique to the tested and stage-3-partnered sessions) by a random function, rendering $\widetilde{tk_{hs}}$, $\widetilde{\mathrm{FS}_{\mathrm{S}}}, \widetilde{\mathrm{FS}_{\mathrm{C}}} \xleftarrow{\$} \{0,1\}^\lambda$, bounded by $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_7}$.

Game 9. Next, we replace HKDF.Expand evaluations keyed with $\widetilde{\mathrm{MS}}$ by a random function, in particular leading to uniformly random values $\widetilde{\mathrm{TS}}, \widetilde{\mathrm{EMS}} \xleftarrow{\$} \{0,1\}^\lambda$ in the tested and partnered sessions, again bounded by $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_8}$. These are moreover independent of any other values computed in sessions not partnered in stages 4 and 5, due to Game 1 and sid_4 and sid_5 fixing the inputs to H_4 and H_5.

Game 10. Finally, we replace the HKDF.Expand evaluations using $\widetilde{\mathrm{TS}}$ (in the tested and partnered sessions) by a random function, resulting in a random application traffic key $\widetilde{tk_{app}} \xleftarrow{\$} \{0,1\}^\lambda$, again bounded by $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_9}$.

In Game 10, all keys derived in the tested session ($\widetilde{tk_{ehs}}$, $\widetilde{tk_{ead}}$, $\widetilde{tk_{hs}}$, $\widetilde{tk_{app}}$, and $\widetilde{\mathrm{EMS}}$) are now chosen uniformly at random, making the Test query independent of the test bit b_{test}. Furthermore, replaying a `ClientHello` to multiple servers leads to all these sessions being partnered (and hence prevents Reveal queries). In contrast, unpartnered sessions (even if using pre-shared secret) use different handshake hashes (due to Game 1) as inputs to HKDF.Expand in the key derivation; resulting keys therefore are uncorrelated with the tested session's keys. □

4.2. PSK-(EC)DHE 0-RTT Handshake

Theorem 4.3 (Match security of draft-14-PSK-DHE-0RTT)**.** *The* draft-14 *PSK-(EC)DHE 0-RTT handshake is* Match*-secure: for any efficient adversary $\mathcal{A}$ we have* $\mathsf{Adv}^{\mathsf{Match}}_{\texttt{draft-14-PSK-DHE-0RTT},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|nonce|}$*, where n_s is the maximum number of sessions, q is the group order, and $|nonce| = 256$ is the bit-length of the nonces.*

Proof (sketch). Beyond the proof of Theorem 4.1, the probability of two sessions picking the same Diffie–Hellman share also enters the bound for session-identifier collisions. □

Theorem 4.4 (Multi-Stage security of draft-14-PSK-DHE-0RTT)**.** *The* draft-14 *PSK-(EC)DHE 0-RTT handshake is* Multi-Stage*-secure in a key-independent and stage-3-forward-secret manner with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *given above. Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_{16}$ such that*

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-14-PSK-DHE-0RTT},\mathcal{A}} \leq 5n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} \\
+ n_p \cdot \Big(& \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3} \\
& + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_5} \Big) \\
+ n_s n_p \cdot \Big(& \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_8} \\
& + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_9} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_{10}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}} \Big) \\
+ n_s n_p \cdot \Big(& \mathsf{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathsf{Extract},\mathbb{G},\mathcal{B}_{12}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_{13}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_{14}} \\
& + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_{15}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_{16}} \Big) \Big),
\end{aligned}$$

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

Proof (abridged). Again we first restrict the adversary $\mathcal{A}$ to a single Test query, inducing a security loss of at most $5n_s$.

Game 1. We next exclude hash collisions, bounded by the advantage $\mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1}$ of a collision reduction $\mathcal{B}_1$.

Case separation. Our proof then treats the following three (disjoint) cases separately:

A. the adversary tests a stage-1 or stage-2 key,
B. the adversary tests a stage-i key for $i \in \{3, 4, 5\}$ in a session without honest contributive partner in stage 3,
C. the adversary tests a stage-i key for $i \in \{3, 4, 5\}$ in a session with honest contributive partner in stage 3.

CASE A. TEST IN STAGE 1–2. As both stages are non–forward-secret, no Corrupt query can have been issued for pss employed in the tested session. This allows the same proof strategy as for Theorem 4.2. Via the very same sequence of games G_2–G_6 we replace both 0-RTT keys tk_{ehs} and tk_{ead} by independent random values (leaving $\mathcal{A}$ no change to win). The introduced differences in advantage of $\mathcal{A}$ are bound as for Theorem 4.2 by $n_p \cdot (\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_2} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_3} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_5})$.

CASE B. TEST IN STAGE 3–5 WITHOUT CONTRIBUTIVE STAGE-3 PARTNER. Since stage 3 is unauthenticated, testing this stage actually leads to immediately losing the Multi-Stage game, hence we can focus on stages 4 and 5. As we will see, given the HMAC values in the exchanged Finished messages are unforgeable, we can ultimately exclude that such Test queries are issued via the following sequence of games.

Game B.1. We first introduce an abortion of the game as soon as a session accepts in stage 4 without honest contributive partner in stage 3. Denoting this event as $\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}$ we can bound the induced advantage difference for $\mathcal{A}$ by $\Pr[\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}]$. Observe that, as Game B.1 aborts before $\mathcal{A}$ has the chance to issue a Test query, we can immediately bound its advantage by 0 in this game and focus on bounding $\Pr[\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}]$ in the remaining game sequence.

Game B.2. Next, we guess the first session that accepts in stage 4 without honest contributive stage-3 partner (i.e., the session causing $\mathsf{abort}^{G_{B.1},\mathcal{A}}_{acc}$), inducing a factor of n_s.

Moreover, no Corrupt query can have been issued to the guessed session (or any other session using the same pre-shared secret pss), as sessions stop execution on corruption of their pre-shared secret and the game aborts when the guessed session accepts in stage 4.

Game B.3. We can now first guess the pre-shared secret $\mathsf{pss} = \mathrm{RMS}$ employed in the guessed session, introducing a factor of at most the number of pre-shared secrets n_p.

Game B.4. Applying to the guessed session the steps introduced in the Games 3, 4, 5, 7, and 8 from the proof of Theorem 4.2, we (in particular) replace the values PSK, ES, HS, HTS, and finally $\mathrm{FS_S}$ and $\mathrm{FS_C}$ by uniformly random values sampled from $\{0,1\}^\lambda$. The introduced advantage difference is bounded by $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_6} + \mathsf{Adv}^{\mathsf{HMAC(0,\$)\text{-}\$}}_{\mathsf{HMAC},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_8} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_9} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{10}}$, where $\mathcal{B}_6$, $\ldots$, $\mathcal{B}_{10}$ are the algorithms $\mathcal{B}_2$, $\mathcal{B}_3$, $\mathcal{B}_4$, $\mathcal{B}_6$, and $\mathcal{B}_7$ given for Games 3, 4, 5, 7, and 8 in the proof of Theorem 4.2.

As the final step, $\mathcal{A}$ triggering $\mathsf{abort}^{G_{B.4},\mathcal{A}}_{acc}$ can be turned into an (EUF-CMA) MAC forger $\mathcal{B}_{11}$ for HMAC. Here, $\mathcal{B}_{11}$ acts as the challenger but computes HMAC values under $\widetilde{\mathrm{FS_S}}$ or $\widetilde{\mathrm{FS_C}}$ through MAC oracles of two EUF-CMA security instances for HMAC. A session accepting in stage 4 (causing $\mathsf{abort}^{G_{B.4},\mathcal{A}}_{acc}$) will receive a MAC (without hash collision by Game 1) which no honest session output, hence a forgery for $\mathcal{B}_{11}$ inducing the final bound $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_{11}}$.

CASE C. TEST IN STAGE 3–5 WITH CONTRIBUTIVE STAGE-3 PARTNER. In this proof case, we leverage the honest Diffie–Hellman shares g^x and g^y (through cid_3-partnering) as source of randomness (unknown to $\mathcal{A}$) which ensures (forward) secrecy of the keys derived in stages 3–5, even if the involved pre-shared secret is corrupted.

Game C.1. We first guess the session contributively partnered with the test session, inducing a factor of n_s.

Game C.2. We guess $\mathsf{pss} = \mathrm{RMS}$, yielding a factor n_p.

Game C.3. We can now encode a Diffie–Hellman challenge in g^x and g^y at the tested session. If a server is tested, the (contributive) partner session might receive a modified $g^{y'}$ for which we need to be able to derive $g^{xy'}$ (without knowing x or y'). To this extent, we model the security of HKDF.Extract deriving HS using ES as salt and $\mathrm{DHE} = g^{xy}$ as source key material using the PRF-ODH assumption [18] (in its single-query variant).

In this game, we replace HS by a uniformly random value $\widetilde{\mathrm{HS}} \xleftarrow{\$} \{0,1\}^\lambda$. In the reduction $\mathcal{B}_{12}$, we employ the challenge Diffie–Hellman values as g^x and g^y in the tested and contributive-partner session, and the PRF challenge as HS. Note that we do not rely on (the secrecy of) ES, which in particular allows pss to be corrupted (later) in the tested session, ensuring forward secrecy (from stage 3 on).

We complete this proof case by applying to the tested session (and its potential partner) the steps described for the Games 7, 8, 9, and 10 from the proof of Theorem 4.2, in particular replacing tk_{hs}, tk_{app}, and EMS by random values, leaving $\mathcal{A}$ no chance to win. This final bound is $\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HMAC},\mathcal{B}_{13}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{15}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{16}}$ where $\mathcal{B}_{13}, \ldots, \mathcal{B}_{16}$ are the algorithms $\mathcal{B}_6$, $\mathcal{B}_7$, $\mathcal{B}_8$, and $\mathcal{B}_9$ given for Theorem 4.2. □

5. The TLS 1.3 draft-12 (EC)DHE 0-RTT Handshake Protocol and its Security

The latest TLS 1.3 drafts do not specify a Diffie–Hellman-based ((EC)DHE) 0-RTT handshake anymore; the last draft doing so is `draft-12` [34]. We nevertheless provide a security analysis of this 0-RTT mode (as specified in `draft-12`) for two reasons: For one, it is much closer to the QUIC and OPTLS protocols and our analysis hence enables a comparison with those designs. For another, it provides slightly stronger forward secrecy properties [21] as reflected in our analysis and may (for that or other reasons) be re-established as a TLS 1.3 extension [38]. Due to space limitations, we only provide a summary of our results on the `draft-12` (EC)DHE 0-RTT handshake here and refer to the full version [14] for the formal definitions.

5.1. Protocol Overview

On a high level, the (EC)DHE 0-RTT handshake goes through the same four phases as the PSK-based 0-RTT modes: key exchange, 0-RTT, server parameters, and authentication (cf. Section 4). A notable difference though is that the client may perform signature-based authentication in the 0-RTT step by sending a certificate and signature on the transcript (in messages $\texttt{ClientCertificate}_0$ resp. $\texttt{ClientCertificateVerify}_0$). Furthermore, authentication is (as in the full (EC)DHE handshake [10], [12]) based on signatures instead of the `Finished` MACs. To enable 0-RTT, servers send a signed `ServerConfiguration` message (in the authentication phase) containing an identifier `config_id` and a semi-static Diffie–Hellman share g^s which the client can use (and indicate in `ClientEarlyData`) for a 0-RTT handshake.

The key schedule of the `draft-12` (EC)DHE 0-RTT handshake is significantly different from that in `draft-14` and works as follows. After sending its `ClientHello` and extensions, the client can already derive one of the two main secret inputs for key derivation, the static secret SS, as the Diffie–Hellman shared value g^{xs}. The initial 0-RTT keys tk_{ehs} and tk_{ead} are then expanded from an intermediate value xSS extracted from SS using HKDF [20] (cf. Section 3 for the HKDF notation). As in `draft-14`, tk_{ehs} is used to encrypt the 0-RTT handshake messages while 0-RTT application data is encrypted with tk_{ead}.

Receiving `ClientHello`, the server derives the 0-RTT keys using its stored configuration for the semi-static key g^s and sends its `ServerHello` message. Both parties then compute the second secret input, the ephemeral secret ES, as the shared Diffie–Hellman value g^{xy}, and an extracted value xES from which the (unauthenticated) handshake traffic key tk_{hs} is expanded. After sending resp. receiving `ServerCertificateVerify` (the server's signature on the transcript), the master secret MS is derived, extracted from intermediate expanded versions mES and mSS of xES and xSS. Both the client and server finished secret $\mathrm{FS_C}/\mathrm{FS_S}$ are derived from MS. Finally, the three final keys are derived from the master secret through HKDF expansion steps: the application traffic key tk_{app}, the resumption master secret RMS (for later preshared-key–based session resumption), and the exporter master secret EMS.

5.2. Security Summary

In a nutshell, the 6-stage `draft-12` (EC)DHE 0-RTT handshake (`draft-12-(EC)DHE-0RTT`) provides unilateral authentication for stage 1 (deriving tk_{ehs}), unilateral or mutual authentication for stage 2 (tk_{ead}), no authentication for stage 3 (tk_{hs}), and one of no, unilateral, or mutual authentication for stages 4–6 (tk_{app}, RMS, EMS). The (0-RTT and regular) handshake keys tk_{ehs} and tk_{hs} are the only ones used internally, and (expectedly) the 0-RTT stages 1–2 are replayable. All keys enjoy key independence and forward secrecy (wrt. compromise of long-term (signing) secrets). Recall that our model treats compromises of long-term and semi-static secrets independently through the Corrupt resp. RevealSemiStaticKey query. While the 0-RTT keys tk_{ehs} and tk_{ead} remain (forward) secret after a long-term key compromise, they are replayable and hence become insecure when the involved semi-static key is revealed. We capture session and contributive identifiers in the same spirit as for the PSK-based modes (cf. Section 4) and model exposure of the semi-static keys g^s and s through the RevealSemiStaticKey query.

In the public-key (MSKE) model we then establish Match security (with a bound of $n_s^2 \cdot 1/q \cdot 2^{-|nonce|}$) and the following Multi-Stage security result for the `draft-12` (EC)DHE 0-RTT handshake.

Theorem 5.1 (Multi-Stage security of `draft-12-(EC)DHE-0RTT`)**.** *The* `draft-12` *(EC)DHE 0-RTT handshake is* Multi-Stage-*secure in a key-independent and stage-1-forward-secret manner with the protocol-specific properties described above. Formally, for any efficient adversary $\mathcal{A}$ against the* Multi-Stage *security there exist efficient algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_{14}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\texttt{draft-12-(EC)DHE-0RTT},\mathcal{A}} \leq 6n_s \cdot \Big(& \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_1} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_2} \\ & + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_3} + n_s \cdot n_{ss} \cdot \Big(\mathsf{Adv}^{\mathsf{msPRF\text{-}ODH}}_{\mathsf{Extract},\mathbb{G},\mathcal{B}_4} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_5} \Big) \\ & + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_6} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_7} + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_8} + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathcal{B}_9} \\ & + \mathsf{Adv}^{\mathsf{COLL}}_{\mathsf{H},\mathcal{B}_{10}} + n_s \cdot \Big(\mathsf{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathsf{Extract},\mathbb{G},\mathcal{B}_{11}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_{12}} \\ & \qquad + \mathsf{Adv}^{\mathsf{st\text{-}Extract}}_{\mathsf{Extract},\mathcal{B}_{13}} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{Expand},\mathcal{B}_{14}} \Big) \Big),\end{aligned}$$

where n_u is the maximum number of users, n_s is the maximum number of sessions, and n_{ss} is the maximum number of semi-static keys.

Proof (sketch). After focusing on a single Test query, our proof separates tests of 0-RTT keys (stage 1–2) and regular keys (stage 3–6). The analysis of the latter case follows closely the one for the `draft-10` full (EC)DHE handshake [12], confirming similar security guarantees hold for the regular keys when adding 0-RTT. As the only change in the advantage bounds, we model the Extract step deriving MS as a strong extractor (instead of a secure PRF) with mES as entropy source and mSS as (public) seed.

The former case can be further split into the tested session not having resp. having a stage-1 contributive partner. For the first sub-case, the tested (necessarily server) session must receive a forged 0-RTT client signature, which can be bounded by the hash function's collision resistance and the signature scheme's unforgeability. In the second sub-case, a Diffie–Hellman challenge can be encoded in g^x and g^s and the derived SS; from the latter (now random) value both 0-RTT keys are expanded randomly (given HKDF.Expand is PRF-secure).

Notably, when encoding the Diffie–Hellman challenge, the reduction must be able to further compute both DH values $g^{x's}$ (for multiple, unknown x') as g^s can be used in multiple server sessions, as well as $g^{xy'}$ (for one, unknown y') in case the server's ephemeral DH share is modified by the adversary in transport. To this extent, we need to employ (on HKDF.Extract) a specific, double-sided variant of the PRF-ODH assumption [18], denoted msPRF-ODH and formally defined in Appendix A, which besides allowing multiple queries on one DH share (as in [22]) additionally allows a single query on the other share. □

6. Comparing the QUIC and TLS 1.3 0-RTT Handshakes

We emphasize two aspects here in which the TLS 1.3 design is superior to QUIC and strengthens the achievable (multi-stage) security both in terms of key independence and compositionality: For one thing, it derives separate keys for the different purposes (in particular, tk_{ehs} and tk_{ead} as well as tk_{hs} and tk_{app} for the encryption of (0-RTT resp. regular) handshake messages and data), enabling a cleaner key separation. For another thing, it establishes authenticity of the server's Diffie–Hellman share g^y through an explicit MAC (PSK-(EC)DHE 0-RTT) resp. signature ((EC)DHE 0-RTT) instead of through an authenticated encryption (under the 0-RTT key) in the data channel, rendering the security of one session key not relying on the secrecy of another.

Conversely, QUIC in its original version `Rev 20130620` achieves replay protection for the derived 0-RTT key on the key exchange level whereas TLS 1.3 does not (and hence, technically, TLS 1.3 satisfies only a weaker notion of security in that respect). As discussed in the beginning, this protection however may become void in the overall setting of secure channels when clients actively replay rejected 0-RTT data over the main channel.

Finally, the (abandoned) Diffie–Hellman-based and the (remaining) PSK-based 0-RTT handshakes in TLS 1.3 (as specified for `draft-12` resp. `draft-14`) differ in the forward-secrecy guarantees they provide for 0-RTT keys, as already pointed out by Krawczyk on the TLS mailing list [21]. While in `draft-12` (EC)DHE 0-RTT those keys are forward secret (wrt. long-term (signing) key compromise) and succumb only to exposures of the semi-static key involved, no forward secrecy is provided in the PSK and PSK-(EC)DHE 0-RTT mode of `draft-14`. It is important to note, though, that preshared resumption secrets used in the PSK-based 0-RTT modes (treated as long-term secrets in our model) are usually much shorter-lived than public-key long-term signing keys, mitigating the effects of a compromise. Still, preshared keys have to be stored safely by both the server and the client—a challenging task in practice, especially on the client's side. Diffie–Hellman-based 0-RTT hence poses weaker requirements in that respect as the client here only has to store the public part of a semi-static key.

7. Composition

Key exchange protocols would be of limited use if applied in isolation; in general the derived keys are meant to be deployed in a follow-up (or overall) protocol. The most common application is of course the encryption (and authentication) of data sent between the two involved parties within a (cryptographic) channel protocol, with the TLS record protocol being a prime example. The TLS 1.3 handshakes (with or without 0-RTT) additionally derive further keys for different purposes, namely the resumption master secret RMS (in non-PSK modes) enabling follow-up abbreviated (preshared-key) handshakes and the exporter master secret EMS which can be used to derive additional key material. For both, the key usage in the cryptographic channel as well as the usage for other purposes, it is desirable to modularize the analysis, treating key exchange and the composed protocol(s) independently and devising automatically the security of the combined execution.

The approach inspires studying the generic compositional guarantees the TLS 1.3 handshake or, in general, a (multi-stage) key exchange protocol can provide. For classical (non–multi-stage) key exchange protocols in the Bellare–Rogaway model [2] this has been argued formally by Brzuska et al. [5], and lifted to the multi-stage setting by Fischlin and Günther [13]. Intuitively, their composition theorem attests that keys derived in a secure (multi-stage) key exchange protocol KE (satisfying certain additional conditions) can be securely used within *any* symmetric-key protocol Π. This result in particular subsumes usage in an arbitrary channel protocol. But, in case of TLS 1.3, it can also be used to argue security of using the resumption master secret established in a full handshake as pre-shared key for a later abbreviated handshake, as done by Dowling et al. [10], [12]. Here, security of the composed protocol $\mathsf{KE}_i;\Pi$ is intuitively defined as the symmetric-key protocol Π being secure when using the stage-i keys established in the key exchange protocol KE (see [13], [10], [12] for a formal definition).

We augment the multi-stage composition result by Fischlin and Günther [13] and extended to—in particular—the preshared-secret setting and multiple concurrent authentication modes by Dowling et al. [10], [12], in order to also capture replayability of keys—which are not generically composable—based on our extended multi-stage key exchange model. The distinction between internal and external (usage of) keys furthermore is eminently useful for defining composition, since it elegantly replaces the necessary informal restriction to final keys in prior theorems.

As its original proof [13], [10] applies with marginal changes, we only state the composition theoremhere and and refer to the full version [14] for a brief discussion of the necessary modifications to the proof.

Theorem 7.1 (Multi-stage composition)**.** *Let* KE *be a* $\mathsf{Multi\text{-}Stage}$*-secure key exchange protocol (in the public-key or preshared-secret setting) providing key independence and stage-j forward secrecy with properties* $(\mathsf{M}, \mathsf{AUTH}, \mathsf{USE}, \mathsf{REPLAY})$ *and key distribution $\mathcal{D}$, and that allows for efficient multi-stage session matching*[10]*. Let Π be a symmetric-key protocol that is secure w.r.t. some game G_Π and has a key generation algorithm that outputs keys with distribution $\mathcal{D}$. Then the composition $\mathsf{KE}_i;\Pi$ for any external and non-replayable stage $i \geq j$ (i.e.,* $\mathsf{REPLAY}_i = \mathsf{nonreplayable}$ *and* $\mathsf{USE}_i = \mathsf{external}$*) 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.*

10. Multi-stage session matching is a technical notion which essentially requires that it is publicly decidable from the transcript whether two sessions are partnered or not when given all keys derived so far (see [11] for a formal definition).

Acknowledgments

We thank the anonymous reviewers for valuable comments. We thank Markulf Kohlweiss for insightful discussions on the necessity of the PRF-ODH assumption for proofs of the TLS 1.3 handshakes. This work has been co-funded by the DFG as part of project S4 within the CRC 1119 CROSSING.

References

[1] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Heidelberg, Aug. 1996.

[2] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, Aug. 1994.

[3] S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman key agreement protocols (invited talk). In S. E. Tavares and H. Meijer, editors, *SAC 1998*, volume 1556 of *LNCS*, pages 339–361. Springer, Heidelberg, Aug. 1999.

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

[5] C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *ACM CCS 11*, pages 51–62. ACM Press, Oct. 2011.

[6] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001.

[7] R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 143–161. Springer, Heidelberg, Aug. 2002. http://eprint.iacr.org/2002/120/.

[8] D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. *Journal of Cryptology*, 22(4):470–504, Oct. 2009.

[9] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe. Automated verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *2016 IEEE Symposium on Security and Privacy*, 2016.

[10] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 1197–1210. ACM Press, Oct. 2015.

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

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

[13] M. Fischlin and F. Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 14*, pages 1193–1204. ACM Press, Nov. 2014.

[14] M. Fischlin and F. Günther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. Cryptology ePrint Archive, Report 2017/082, 2017. http://eprint.iacr.org/2017/082.

[15] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. Non-interactive key exchange. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, Heidelberg, Feb. / Mar. 2013.

[16] B. Hale, T. Jager, S. Lauer, and J. Schwenk. Speeding: On low-latency key exchange. Cryptology ePrint Archive, Report 2015/1214, 2015. http://eprint.iacr.org/2015/1214.

[17] S. Halevi and H. Krawczyk. One-pass HMQV and asymmetric key-wrapping. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 317–334. Springer, Heidelberg, Mar. 2011.

[18] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, Heidelberg, Aug. 2012.

[19] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, Aug. 2005.

[20] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Heidelberg, Aug. 2010.

[21] H. Krawczyk. [TLS] Call for consensus: Removing DHE-based 0-RTT. https://mailarchive.ietf.org/arch/msg/tls/xmnvrKEQkEbD-u8HTeQkyitmclY, Mar. 2016. posting in above thread.

[22] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448. Springer, Heidelberg, Aug. 2013.

[23] H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978, 2015. http://eprint.iacr.org/2015/978.

[24] H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. In *2016 IEEE European Symposium on Security and Privacy*, pages 81–96. IEEE, Mar. 2016.

[25] A. Langley and W.-T. Chang. QUIC Crypto, June 2013. Revision 20130620.

[26] A. Langley and W.-T. Chang. QUIC Crypto. https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/, July 2015. Revision 20150720.

[27] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. How secure and quick is QUIC? Provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy*, pages 214–231. IEEE Computer Society Press, May 2015.

[28] K. G. Paterson and T. van der Merwe. Reactive and proactive standardisation of TLS. In L. Chen, D. McGrew, and C. Mitchell, editors, *SSR 2016*, volume 10074 of *Lecture Notes in Computer Science*, pages 160–186. Springer, Dec. 2016.

[29] W. M. Petullo, X. Zhang, J. A. Solworth, D. J. Bernstein, and T. Lange. MinimaLT: minimal-latency networking through better security. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 425–438. ACM Press, Nov. 2013.

[30] QUIC, a multiplexed stream transport over UDP. https://www.chromium.org/quic.

[31] E. Rescorla. 0-RTT and Anti-Replay (IETF TLS working group mailing list). https://www.ietf.org/mail-archive/web/tls/current/msg15594.html, Mar. 2015.

[32] 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, Oct. 2015.

[33] E. Rescorla. Should it be possible to do 0-RTT with the server signing (pull request #443). https://github.com/tlswg/tls13-spec/issues/443, Apr. 2016.

[34] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-12. https://tools.ietf.org/html/draft-ietf-tls-tls13-12, Mar. 2016.

[35] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-13. https://tools.ietf.org/html/draft-ietf-tls-tls13-13, May 2016.

[36] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-14. https://tools.ietf.org/html/draft-ietf-tls-tls13-14, 2016.

[37] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-18. https://tools.ietf.org/html/draft-ietf-tls-tls13-18, Oct. 2016.

[38] E. Rescorla. TLS 1.3 — draft-ietf-tls-tls13-12 (presentation at ietf 95 meeting). https://www.ietf.org/proceedings/95/slides/slides-95-tls-2.pdf, April 2016.

[39] D. J. Wu, A. Taly, A. Shankar, and D. Boneh. Privacy, discovery, and authentication for the internet of things. In I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, and C. A. Meadows, editors, *ESORICS 2016, Part II*, volume 9879 of *LNCS*, pages 301–319. Springer, Heidelberg, Sept. 2016.

Appendix A. The $\mathsf{msPRF\text{-}ODH}$ Assumption

Definition A.1 ($\mathsf{msPRF\text{-}ODH}$ assumption)**.** *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q with generator g, $\mathsf{PRF}\colon \mathbb{G} \times \{0,1\}^* \to \{0,1\}^\lambda$ be a pseudorandom function with keys in $\mathbb{G}$, input strings from $\{0,1\}^*$, and output strings of length λ, let $b \in \{0,1\}$ be a bit, and $\mathcal{A}$ be a PPT algorithm.*

We define the following $\mathsf{msPRF\text{-}ODH}$ *security game $G^{\mathsf{msPRF\text{-}ODH},b}_{\mathsf{PRF},\mathbb{G},\mathcal{A}}$:*

Setup. *The challenger chooses $v \xleftarrow{\$} \mathbb{Z}_q$ at random and gives g^v to $\mathcal{A}$.*

Query 1. *In the next phase $\mathcal{A}$ can ask queries of the form $(g^u, x) \in (\mathbb{G}, \{0,1\}^*)$ which the challenger answers with the value $y \leftarrow \mathsf{PRF}((g^u)^v, x)$.*[11]

Challenge. *At some point $\mathcal{A}$ asks a challenge query $\hat{x} \in \{0,1\}^*$ on which the challenger chooses $\hat{u} \xleftarrow{\$} \mathbb{Z}_q$ at random, sets $\hat{y}_0 \leftarrow \mathsf{PRF}(g^{\hat{u}v}, \hat{x})$ and $\hat{y}_1 \xleftarrow{\$} \{0,1\}^\lambda$, and answers with $(g^{\hat{u}}, \hat{y}_b)$.*

Query 2. *Again, $\mathcal{A}$ can ask queries of the form $(g^u, x) \in (\mathbb{G}, \{0,1\}^*)$ which the challenger answers with the value $y \leftarrow \mathsf{PRF}((g^u)^v, x)$, except that $\mathcal{A}$ is not allowed to query the pair $(g^{\hat{u}}, \hat{x})$.*

Additionally, and this is where our version differs from the previous $\mathsf{PRF\text{-}ODH}$ *assumption, adversary $\mathcal{A}$ can ask one distinct query of the form $(g^{\hat{v}}, x) \in (\mathbb{G}, \{0,1\}^*)$ for $g^{\hat{v}} \neq g^v$ which the challenger answers with the value $y \leftarrow \mathsf{PRF}((g^{\hat{v}})^{\hat{u}}, x)$.*

Guess. *Eventually, $\mathcal{A}$ stops and outputs a bit b' which is also the game output, denoted by $G^{\mathsf{msPRF\text{-}ODH},b}_{\mathsf{PRF},\mathbb{G},\mathcal{A}}$.*

We define the advantage function $\mathsf{Adv}^{\mathsf{msPRF\text{-}ODH}}_{\mathsf{PRF},\mathbb{G},\mathcal{A}} := \left| \Pr\left[G^{\mathsf{msPRF\text{-}ODH},0}_{\mathsf{PRF},\mathbb{G},\mathcal{A}} = 1\right] - \Pr\left[G^{\mathsf{msPRF\text{-}ODH},1}_{\mathsf{PRF},\mathbb{G},\mathcal{A}} = 1\right] \right|$ *and, assuming a sequence of groups in dependency of the security parameter, we say that the* $\mathsf{msPRF\text{-}ODH}$ *assumption holds for* PRF *with keys from $(\mathbb{G}_\lambda)_\lambda$ if for any $\mathcal{A}$ the advantage function is negligible (as a function in λ).*

11. We require that the first element is in $\mathbb{G}$ and hence write it as g^u, although $\mathcal{A}$ does not necessarily know u.