KEMTLS with Delayed Forward Identity Protection in (Almost) a Single Round Trip

Felix Günther[1], Simon Rastikian[1,2,⋆], Patrick Towa[1] and Thom Wiggers[3]

[1] ETH Zurich, Zurich, Switzerland
mail@felixguenther.info, patrick.towa@inf.ethz.ch
[2] IBM Research Europe, Zurich, Switzerland
sra@zurich.ibm.com
[3] Radboud University, Nijmegen, Netherlands
thom@thomwiggers.nl

Abstract. The recent KEMTLS protocol (Schwabe, Stebila and Wiggers, CCS'20) is a promising design for a quantum-safe TLS handshake protocol. Focused on the web setting, wherein clients learn server public-key certificates only during connection establishment, a drawback of KEMTLS compared to TLS 1.3 is that it introduces an additional round trip before the server can send data, and an extra one for the client as well in the case of mutual authentication. In many scenarios, including IoT and embedded settings, client devices may however have the targeted server certificate pre-loaded, so that such performance penalty seems unnecessarily restrictive.
This work proposes a variant of KEMTLS tailored to such scenarios. Our protocol leverages the fact that clients know the server public keys in advance to decrease handshake latency while protecting client identities. It combines medium-lived with long-term server public keys to enable a delayed form of forward secrecy even from the first data flow on, and full forward secrecy upon the first round trip. The new protocol is proved to achieve strong security guarantees, based on the security of the underlying building blocks, in a new model for multi-stage key exchange with medium-lived keys.

Keywords: Key Exchange, Post-Quantum, Identity Protection, KEMTLS

1 Introduction

The Transport Layer Security (TLS) protocol is among the most widely deployed cryptographic protocols. It is used to securely access web pages, email servers, Internet-of-Things (IoT) gateways or even servers in Cooperative Intelligent Transport Systems [48] (C-ITSs). In the TLS *handshake* sub-protocol, a client and a server authenticate each other (at least the server to the client) and

⋆ Part of the work was completed while the second author was affiliated with DIENS, École Normale Supérieure, PSL University, Paris, France.

jointly establish a symmetric key that is then used in the *record* sub-protocol to privately communicate authenticated application data. The latest version of the protocol, standardized in 2018, is TLS 1.3 [43] and uses an ephemeral Diffie–Hellman key exchange to establish keys that remain secure even after a potential compromise of the parties' long-term keys, i.e., enabling so-called *forward secrecy*.

Post-Quantum TLS. In anticipation of large-scale quantum computers, several candidates for a post-quantum version of the TLS handshake protocol have emerged. These for instance include the CECPQ2 experiment [36,38] by Google that combines X25519 ECDH with the NTRU-HRSS lattice-based key exchange in the TLS 1.3 handshake, or the Open Quantum Safe initiative [50] with prototype integrations in the OpenSSL library of TLS 1.3 key exchange with hybrid security.

A promising candidate in this area is the KEMTLS protocol recently proposed by Schwabe, Stebila, and Wiggers [46]. It is free of handshake signatures and only relies on key encapsulation to provide both key establishment and authentication in a quantum-safe way. The main idea is reminiscent of the OPTLS protocol [34] (which in turn inspired the TLS 1.3 handshake design): at its core are encapsulations against the respective partner's public key, using the resulting secrets to establish a shared key. As the resulting shared key can only be recovered with the partner's secret key, this approach implicitly authenticates the partner. Besides, to enable forward secrecy, the client also sends at the beginning of the protocol an ephemeral public key that the server encapsulates against to obtain an ephemeral contribution. The prototype implementation of KEMTLS showed that its bandwidth was over 50% lighter than that of a size-optimized post-quantum instantiation of TLS 1.3, and that it reduces the amount of CPU cycles by almost 90% compared to a speed-optimized post-quantum instantiation of TLS 1.3.

However, the KEMTLS protocol only treats the classical web scenario in which the client has no prior knowledge of the server public key, although the client could in practice cache the server certificate during an initial handshake. In IoT or embedded-device settings, the server public key is often even hardcoded, e.g., in firmware. The client therefore knows the server public key ahead of time in many practical scenarios. This knowledge not only has the benefit of allowing the client to verify the server certificate only once before any handshake (thereby speeding handshakes and saving power for IoT devices), but could potentially lead to a protocol with fewer message round trips, which is in practice crucial to reduce network latency (also in the web setting) and power consumption.

Indeed, in the KEMTLS protocol, the server cannot send application data before the client does, and the client can only send data after two round trips in the case of mutual authentication, i.e., it is a two-Round-Trip-Time or 2-RTT protocol. This contrasts with TLS 1.3, where the server can send data (e.g., a server banner or an IoT-hub certificate) from its first message flow and the client can do so after a single round trip even in the case of mutual authentication.

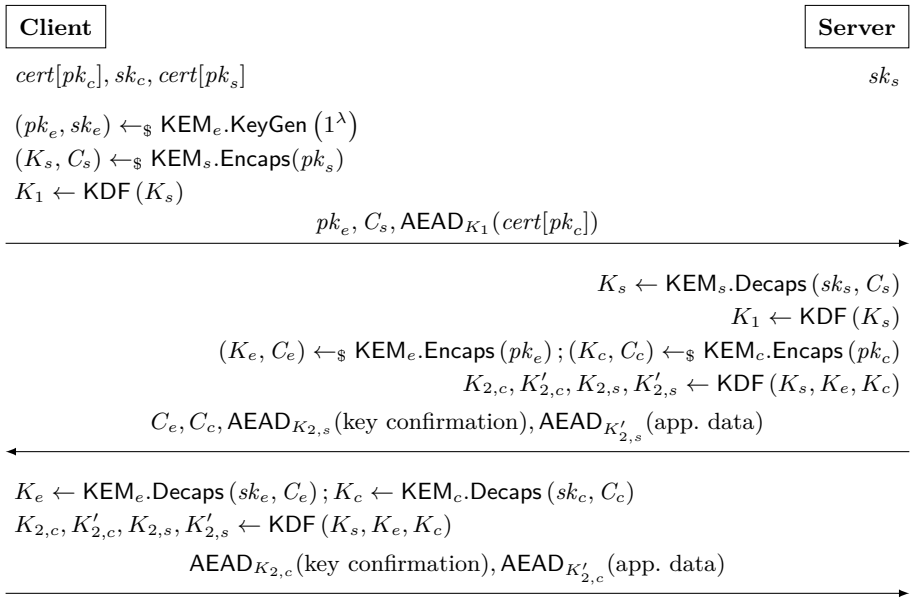

Fig. 1: A 1-RTT protocol without forward identity protection.

The underlying reason is that each party must wait for the other's public key to then encapsulate against it, thereby implicitly authenticating the latter, since there are no handshake signatures as in TLS 1.3. Scenarios in which the client knows the server public key from the beginning of the protocol hence promise to enable substantial performance improvements.

No Forward Identity Protection in 1-RTT. In case the client must also authenticate herself to the server, as it is for instance necessary for IoT devices or vehicles in C-ITSs, the TLS protocol is expected to also provide *identity protection* [32], namely that the client's identity should only be recoverable by a server that is already authenticated. The client can of course leverage the server public key that it already knows to encrypt her certificate (as illustrated in Figure 1), but since there is no ephemeral contribution from the server yet, an adversary that compromises the server's key could recover the client's identity even *after* the handshake completed. In other words, there would be no *forward(-secure)* identity protection.

Despite the efficiency benefits of a 1-RTT protocol, forgoing forward identity protection altogether might be too great of a compromise, especially when privacy is a primary concern. For instance, the European Telecommunications Standards Institute identifies the high risk of user profiling as a main privacy challenge in IoT [21]. The US National Institute of Standards and Technology considers as a high-level risk mitigation "safeguarding the confidentiality [...] of data [...] collected by, stored on, processed by, or transmitted to or from

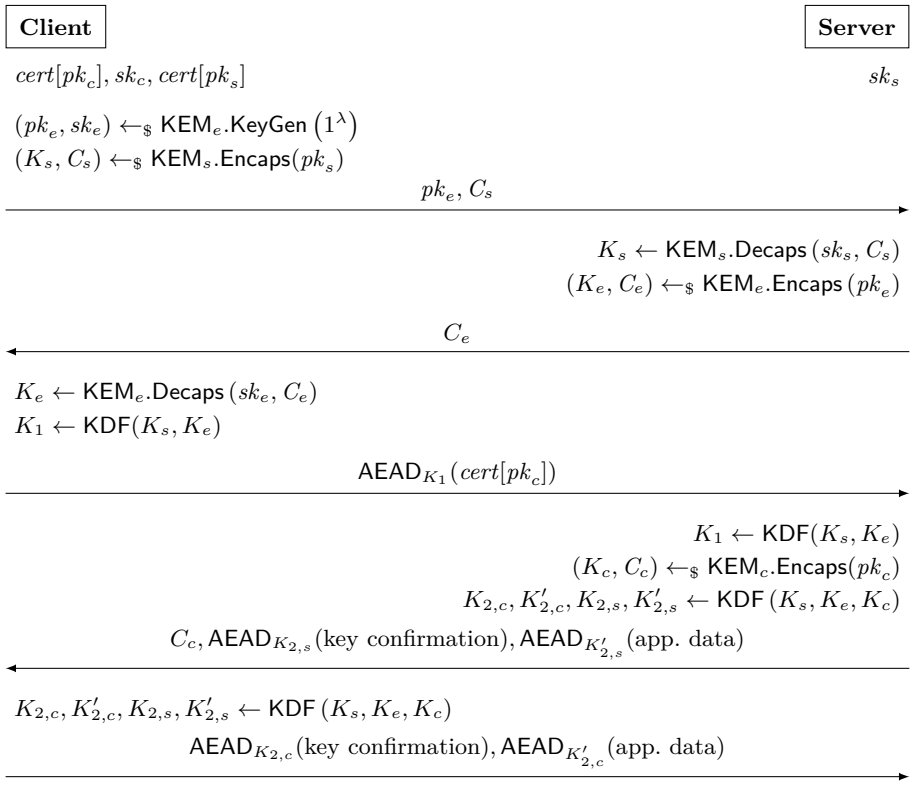

Fig. 2: A 2-RTT protocol with forward identity protection.

the IoT device" [22] and stated that an IoT device should have "the ability to use demonstrably secure cryptographic modules for standardized cryptographic algorithms [...] to prevent the confidentiality [...] of the device's stored and transmitted data from being compromised" [23]; here, the client identity belongs to such transmitted data.

Nevertheless, to maintain client privacy (in a protocol using only key encapsulation) even if the server long-term keys are later compromised, the client cannot send her certificate before the server has made an ephemeral contribution in a first round trip. This means the client cannot be authenticated before the server encapsulates against her public key in a second round trip (see Figure 2). There seems to be no way of fully leveraging the knowledge of the server public key to have a 1-RTT protocol while maintaining forward identity protection.

1.1 Contributions

The core contribution of this paper is a protocol (in Section 3) that bridges the gap between forward identity protection and a 1-RTT protocol solely based

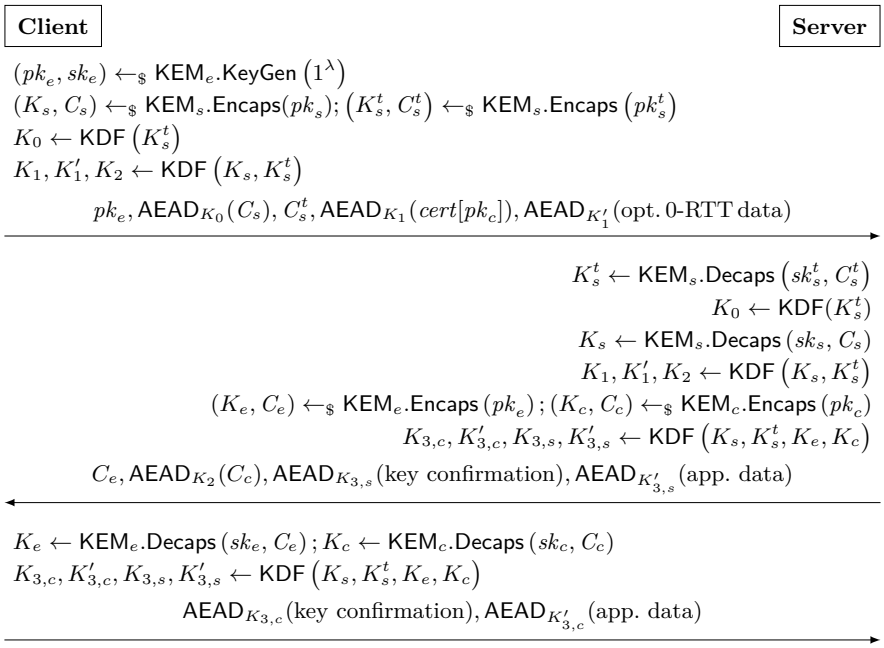

Fig. 3: Sketch of the main protocol.

on key encapsulation, under the assumption that the client knows the server public key at the start of the protocol (see Figure 3 for a sketch). The main idea is to introduce semi-static public keys on the server side which the client also knows at the start of the protocol. These semi-static keys are periodically refreshed (e.g., once every other day), and if the corresponding secret key is not compromised before it expires, the client's identity can no longer be recovered, even if the server long-term secret key is later compromised. In this sense, the protocol satisfies a *delayed* form of forward identity protection [9] without any extra round compared to a 1-RTT protocol without forward identity protection. As a side-effect, the protocol also allows for optional zero round-trip time (0-RTT) data with the same delayed forward secrecy, which the client can already send within its first flight without having to wait for the server. Since the semi-static keys are not assumed to be certified (the protocol would otherwise be impractical), they must be transmitted during an initial handshake that then consists of two round trips. The protocol takes care of this mechanism, and allows for semi-static keys to roll over between two time periods, so that servers can serve clients using both the key for the current and the next time periods.

Section 4 presents a model that formalizes the properties expected from a protocol involving semi-static keys, and Section 5 proves (in the reductionist framework and in exact-security terms) that the protocol does satisfy them under standard assumptions. The model in Section 4 is closely related to the multi-stage

key exchange model [24] proposed for TLS 1.3 [18,19] and that for KEMTLS [46], but it also accounts for the semi-static keys and their lifetime. Section 5 then shows that the protocol achieves the intended security levels across the various stages of the handshake, relying only on standard-model assumptions. Section 6 compares the protocol to alternative approaches and highlights its advantages. Section 7 discusses implementation choices as well as a prototype implementation and Section 8 discusses benchmarking results. As expected, caching certificates incurs significant performance gains as it reduces the handshake time by at least 44%, and the privacy gains from semi-static keys come at negligible performance costs.

Concurrent work. In concurrent work, Schwabe, Stebila, and Wiggers [47] also consider a variant of the KEMTLS protocol, called KEMTLS-PDK, that leverages prior knowledge of peer public keys. Similarly to this work, they show how pre-distributed public keys can lead to reduced round trips and bandwidth for the handshake (as on Figure 1). Their work further explores the performance characteristics for various NIST post-quantum KEM candidates. This work in contrast focuses on identity privacy and forward secrecy: beyond leveraging pre-distributed long-term keys, our protocol additionally employs in-band-distributed, semi-static keys to achieve (delayed) forward secrecy for the first data flow including the client's identity and, optionally, 0-RTT data.

2 Preliminaries

This section introduces the notation used throughout the paper and the cryptographic primitives on which the protocols herein rely.

2.1 Notation

The security parameter is denoted λ and is encoded in unary when given as input to algorithms. For an integer $n \geq 1$, $[\![n]\!]$ denotes the set $\{1, \ldots, n\}$. The notation $y \leftarrow \mathsf{A}(x)$ or $\mathsf{A}(x) \rightarrow y$ means that a deterministic algorithm A runs on input x and returns y; for probabilistic algorithms the notation $\leftarrow_{\$}$ resp. $\rightarrow_{\$}$ is used instead.

2.2 Hash Functions

A hash function $H\colon \mathcal{X} \rightarrow \{0,1\}^{\ell}$ is a map from a potentially infinite set $\mathcal{X}$ to the set of bit strings of a fixed length $\ell(\lambda)$. The advantage of an adversary $\mathscr{A}$ in finding a collision for H is defined as $\Pr\left[x \neq y \wedge H(x) = H(y)\colon (x,y) \leftarrow_{\$} \mathscr{A}\left(1^{\lambda}\right)\right]$.

2.3 Pseudorandom Functions

A pseudorandom function (PRF) [26] is an efficiently computable function with values computationally indistinguishable from uniformly random values.

Formally, a function $\mathsf{PRF}\colon \mathcal{K}\times\mathcal{X}\to\mathcal{Y}$ is a (T,q,ε)-secure PRF with key space $\mathcal{K}$, input space $\mathcal{X}$ and range $\mathcal{Y}$ (all assumed to be finite and of size depending on a security parameter λ) if the advantage

$$\left|\Pr\left[1\leftarrow\mathscr{A}^{\mathsf{PRF}(K,\cdot)}\colon K\leftarrow_{\$}\mathcal{K}\right]-\Pr\left[1\leftarrow\mathscr{A}^{R(\cdot)}\colon R\leftarrow_{\$}\mathcal{Y}^{\mathcal{X}}\right]\right|$$

of every adversary $\mathscr{A}$ that runs in time at most $T(\lambda)$ and makes at most $q(\lambda)$ queries is at most $\varepsilon(\lambda)$.

A function PRF is a (T,q,ε)-secure *dual* PRF [3] if $\mathsf{PRF}'\colon (x,y)\mapsto\mathsf{PRF}(y,x)$ is a (T,q,ε)-secure PRF.

2.4 Key Derivation Functions

A key derivation function (KDF) is an algorithm which computes pseudorandom keys of appropriate length from a source key material which is not necessarily uniformly distributed, but still has high entropy despite potential partial adversarial knowledge. The results can then be used as secret keys for cryptosystems.

Syntax. A key derivation function [33] $\mathsf{KDF}(XTS,SKM,CTX,L)\to K$ takes as input an extractor-salt value XTS, a source key material SKM, some context information CTX and a length L, and returns an L-bit string K.

Hash-Based Key Derivation. Krawczyk [33] proposed a hash-bashed KDF, denoted HKDF, that follows the extract-then-expand paradigm. It consists of two functionalities, namely an $\mathsf{Extract}(XTS,SKM)\to\text{PRK}$ algorithm that computes a key for pseudo-random evaluation from the salt and the source key material, and an $\mathsf{Expand}(\text{PRK},CTX,L)\to K$ algorithm that computes an L-bit key via successive pseudo-random evaluations from the pseudo-random key and the context information.

2.5 Message Authentication Codes

A message authentication code (MAC) is a primitive that attests the authenticity of a message using a private key. It consists of an algorithm $\mathsf{KeyGen}\left(1^{\lambda}\right)\to_{\$}K$ that generates a private key and an algorithm $\mathsf{MAC}(K,M)\to\tau$ that computes a tag τ on a message M using a key K. A MAC is considered (T,q,ε)-Existentially UnForgeable under Chosen-Message Attacks (EUF-CMA) if for every algorithm $\mathscr{A}$ that runs in time at most $T(\lambda)$ and makes at most $q(\lambda)$ oracle queries,

$$\Pr\left[\begin{array}{c}\mathsf{MAC}\left(K,\mu\right)=\tau\\ \wedge\mu\notin Q\end{array}\colon\begin{array}{c}K\leftarrow_{\$}\mathsf{KeyGen}\left(1^{\lambda}\right);Q\leftarrow\emptyset\\ (\mu,\tau)\leftarrow_{\$}\mathscr{A}^{\mathscr{O}(K,\cdot)}\end{array}\right]\leq\varepsilon(\lambda),$$

with $\mathscr{O}$ an oracle which, on input K, replies to a query on a message M by computing and returning $\mathsf{MAC}\left(K,M\right)$ and adding M to Q.

2.6 Key Encapsulation Mechanisms

The protocol further relies on key encapsulation mechanisms (KEMs), a public-key primitive which allows a party to send a symmetric key to another party encrypted under the public key of the latter. It consists of a key generation algorithm $\mathsf{KeyGen}\left(1^\lambda\right) \to_\$ (pk, sk)$ that generates a pair of public and secret keys, an encapsulation algorithm $\mathsf{Encaps}(pk) \to_\$ (K, C)$ which computes a symmetric key in a set $\mathcal{K}$ and a ciphertext, and a decapsulation algorithm $\mathsf{Decaps}(sk, C) \to K$ that computes a symmetric key on the input of a secret key and a ciphertext.

A KEM is deemed δ-correct [29] if

$$\Pr\left[K \neq \mathsf{Decaps}(sk, C) \colon \begin{array}{c}(pk, sk) \leftarrow_\$ \mathsf{KeyGen}\left(1^\lambda\right) \\ (K, C) \leftarrow_\$ \mathsf{Encaps}(pk)\end{array}\right] \leq \delta(\lambda).$$

The security of a KEM requires the keys it generates to be indistinguishable from uniformly random values in $\mathcal{K}$, and in certain cases even if an adversary is given access to a decapsulation oracle. If the adversary is given access to such an oracle, the security notion is referred to as INDistinguishability under Chosen-Ciphertext Attacks (or IND-CCA security), and otherwise as INDistinguishability under Chosen-Plaintext Attacks (or IND-CPA security).

Formally, for $atk \in \{\mathrm{CPA}, \mathrm{CCA}\}$, a KEM satisfies (T, q, ε)-IND-atk security if for every adversary $\mathscr{A}$ that runs in time at most $T(\lambda)$ and makes at most $q(\lambda)$ oracle queries,

$$\left|\Pr\left[b = b' \colon \begin{array}{c}(pk^\star, sk^\star) \leftarrow_\$ \mathsf{KeyGen}\left(1^\lambda\right) \\ b \leftarrow_\$ \{0,1\} \\ (K_0^\star, C^\star) \leftarrow_\$ \mathsf{Encaps}\left(pk^\star\right) \\ K_1^\star \leftarrow_\$ \mathcal{K} \\ b' \leftarrow_\$ \mathscr{A}^{\mathscr{O}_{atk}(sk^\star, C^\star, \cdot)}\left(pk^\star, C^\star, K_b^\star\right)\end{array}\right] - 1/2\right| \leq \varepsilon(\lambda),$$

with $\mathscr{O}_{atk}$ an oracle which, on input $C^\star$, to replies to a decapsulation query on a ciphertext C with

- $\mathsf{Decaps}\left(sk^\star, C\right)$ if $C \neq C^\star$ and $atk = \mathrm{CCA}$,
- $\perp$ otherwise.

(T, ε)-IND-1CCA security refers to $(T, 1, \varepsilon)$-IND-CCA security, i.e., a security notion in which the adversary makes at most one decapsulation query.

3 Protocol

This section presents a key-exchange protocol, specified in Figure 4, with mutual authentication that solely relies on KEMs for key establishment and authentication between a client and a server. The protocol assumes the client to have prior knowledge of the server certificate, as it is often the case for embedded or IoT devices and other applications of TLS. This, together with novel insights, allows the client to send forward-secret and fully authenticated application data

after a single round trip, and the server from its first flow, as in TLS 1.3. It also allows (optional) zero round-trip time (0-RTT) to be send by the client along with its first flight of messages. In comparison, in the KEMTLS protocol [46] the client can only send application data after two round trips in the case of mutual authentication, and the server can only do so from its second flow regardless of client authentication.

3.1 Protocol Description

Building Blocks. The protocol involves three KEMs:

- KEM_e for establishing ephemeral secrets and enabling forward secrecy,
- KEM_c for implicit client authentication, and
- KEM_s for implicit server authentication.

All three could be instantiated with the same scheme or be chosen differently depending on various optimization factors. For instance, KEM_e could be chosen so as to minimize the key-generation time and alleviate client computation, whereas KEM_c and KEM_s could be selected as schemes with fast encapsulation even though key generation might be long, with an even stronger computational-efficiency requirement for the client than for the server.

Besides, the protocol also uses

- Krawczyk's hash-based key derivation function HKDF [33] as keystone of the key schedule to extract randomness from the KEM-generated secrets and derive stage keys,
- HMAC [5] as message authentication code for explicit party authentication, and
- a hash function H, e.g., SHA-256, to compute expansion labels for HKDF as well as compress the handshake messages before explicit authentication.

Outline. The protocol shares similarities with the KEMTLS protocol, which is itself modeled after the OPTLS protocol [34]. However, it goes beyond prior work to reconcile client privacy (even if server long-term keys are later compromised) and a 1-RTT handshake: it leverages *server semi-static KEM keys* which the client encapsulates against and mixes the result into the key schedule at the beginning of the protocol, so that only a party privy to the semi-static secret key can decipher the client identity.

Key Lifetime. A pair of semi-static keys is only to be used in a given time period, e.g., a duration of two days, after which the server refreshes the pair. Though the privacy guarantees are not as strong as those of a 2-RTT handshake which uses fully ephemeral secrets to protect client certificates, they are still relevant in practice and it is a fair compromise for the efficiency benefits.

Clocks. The server keeps track of time periods with an integer counter. Only the server must maintain a clock, just to know when to refresh the keys. The client need only store the latest semi-static public key it received from the server along with the corresponding time period, which is indicated by the server. This means that the protocol can even be used with clients that may not have a clock as it is the case for some IoT devices.

Time-Period Transition. The server generates the keys for a time period before its beginning and sends the public key to the client as part of a handshake during a transition phase from the previous time period, e.g., the last hour. During this transition phase, the server not only accepts handshake requests with the current key, but also with the next one, so that the client can use the next key as soon as it receives it.

In case the client does not connect to the server during this transition phase, the client simply initiates the protocol with the latest known key (if any) in addition to the server long-term key. The server then just rejects the ciphertext encrypting the client certificate and returns the current public key; following in spirit the `HelloRetryRequest` message sent in a TLS 1.3 handshake upon configuration mismatch [43, Section 4.1.4]. The client can now send its certificate anew, encrypted under a mixture of ephemeral KEM secret (instead of the skipped semi-static secret) and long-term KEM secret. The two parties otherwise follow essentially the same flow as in Figure 4, leading to only a one-time delay by one round trip to re-synchronize.

Protocol Notation. Table 1 summarizes the notation used for the protocol secrets. In Figures 4 and 5,

- MSG : M denotes that message MSG is sent and contains M.
- $\{\text{MSG}\}_{\text{stage}_k}$: M denotes the AEAD encryption of a message MSG containing M under an AEAD key derived from the secret accepted at stage k (the derivation is not made explicit on the figures). A star (*) as superscript indicates that the message is only sent the during the transition from the current server time period to the next.

Inputs. At the beginning of the protocol, in addition to its certificate $cert[pk_c]$ and secret key sk_c, the client holds a server long-term certificate $cert[pk_s]$ and the latest server semi-static key $pk_s^{t_{s,c}}$ known to the client in a time period $t_{s,c}$. By convention, $pk_s^{t_{s,c}} := \perp$ and $t_{s,c} := -\infty$ if the client has never obtained a semi-static key from the intended partner server. As for the server, it is given as input a long-term secret key sk_s and a semi-static secret $sk_s^{t_s}$ corresponding to the current server time period t_s. Note that the long-term public keys are certificated out of band by an external certification authority. In constrast, the semi-static public keys are *not* assumed to be certified.

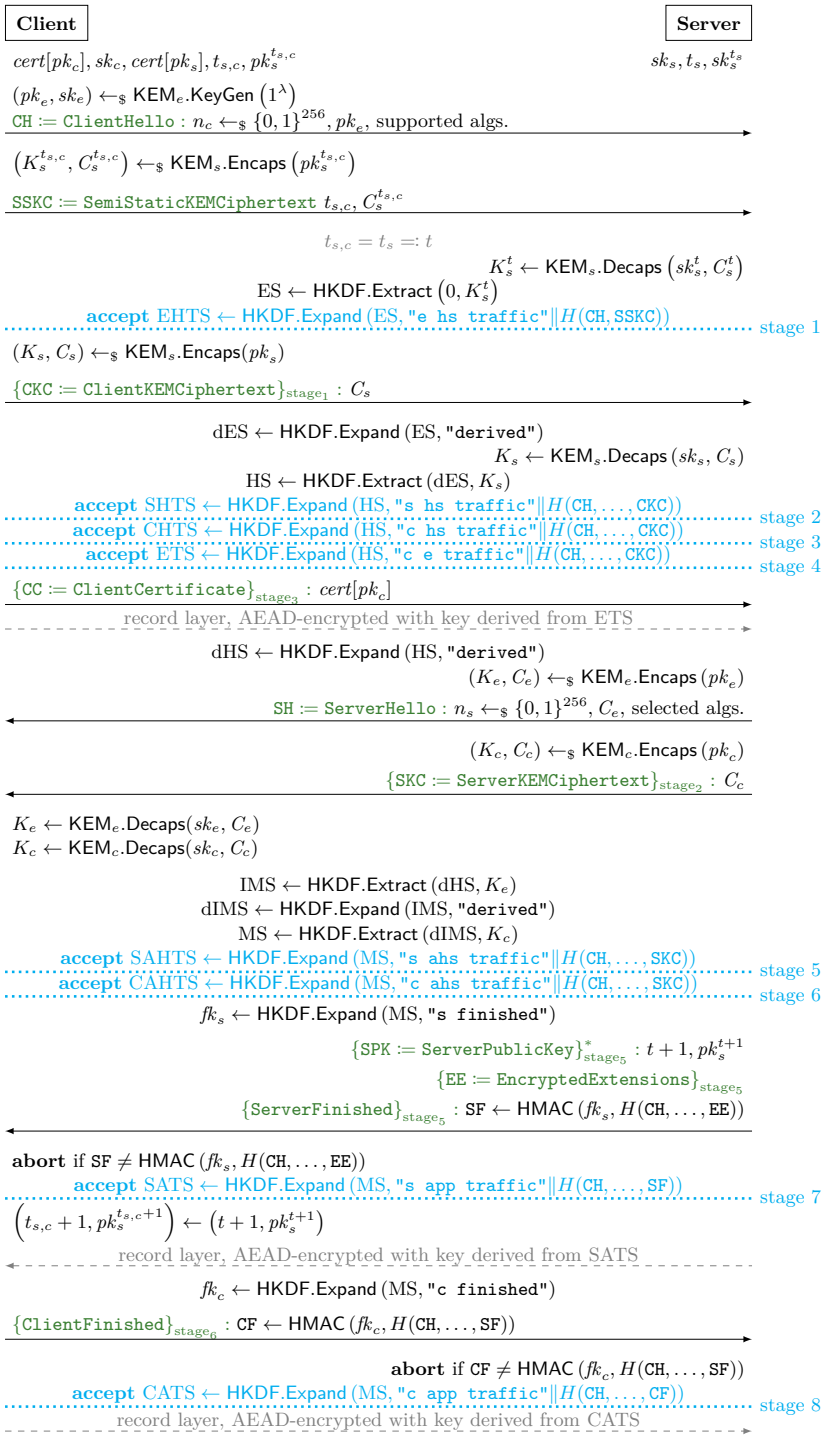

Fig. 4: Protocol in the case of matching time periods.

Fig. 5: Protocol in the case of unmatching time periods.

Protocol Steps. The main protocol steps are as follows.

- The client first generates a pair (pk_e, sk_e) of ephemeral keys, and sends the public key with a fresh nonce n_c and the list of algorithms it supports in a `ClientHello` message.
- The client then encapsulates against the server semi-static key and sends the resulting ciphertext $C_s^{t_{s,c}}$ together with its time period $t_{s,c}$ within a `SemiStaticKEMCiphertext` message. It uses the resulting semi-static secret $K_s^{t_{s,c}}$ to compute an early secret ES, and from that derive a first stage key, the early handshake traffic secret EHTS.
- The client next encapsulates against the server long-term key, sending the resulting ciphertext C_s AEAD-encrypted under EHTS. (This protects the certified identity of the server from an active adversary with delayed forward secrecy, in case C_s may leak such information.) The resulting key K_s is mixed into the key schedule to obtain a handshake secret HS, which implicitly authenticates the server.
- The handshake secret HS is used to compute server and client handshake traffic secrets SHTS and CHTS. The client uses CHTS to AEAD-encrypt its certificate in a `ClientCertificate` message. (This ensures that only a party knowing both the server long-term and semi-static secret keys used can infer information about the client's identity.)
  Additionally, an early traffic secret ETS is derived to optionally send protected 0-RTT application data.
- When the server receives the `ClientHello`, `SemiStaticKEMCiphertext` and `ClientCertificate` messages from the client, two cases arise: either the client time period $t_{s,c}$ matches the current server time period t_s or not.

  Matching time periods. If $t_{s,c} = t_s =: t$ (or $t_{s,c} = t_s + 1$ during the transition from t_s to $t_s + 1$) as in Figure 4, the server has the semi-static secret key sk_s^t and can thus compute EHTS, SHTS, CHTS, and ETS, and recover C_s, the client certificate and potential early application data.
  - The server encapsulates against pk_e and sends in a `ServerHello` message the resulting ciphertext C_e, together with a fresh nonce and the algorithms selected from the algorithms that the client supports.
  - Next, the server encapsulates against the client public key pk_c and encrypts the resulting ciphertext C_c under SHTS. (This prevents information about the client's identity to leak through C_c.)
  - Both parties now compute a master secret by mixing in the ephemeral and client long-term KEM secrets K_e and K_c. Secret K_e enables forward secrecy, K_c implicitly authenticates the client.
  - From MS, both parties compute (mutually) authenticated handshake traffic secrets SAHTS and CAHTS for the server and the client, used to derive AEAD keys to encrypt the remaining handshake.
  - From MS, MAC "finished" keys fk_s and fk_c are further derived for explicit authentication as well as application transport secrets SATS and CATS for application data encryption.

* The server explicitly authenticates by sending a "finished" message, a MAC over the transcript under key fk_s and can then send application data.
  During a transition phase to the next time period, it also sends the public key pk_s^{t+1} for the next time period.[4] The client saves this key (and discards pk_s^t) only after verifying the server MAC.
* Upon receiving the server "finished" message, the client explicitly authenticates by also sending a MAC over the transcript under key fk_c, and can then send application data.

Unmatching time periods. If $t_{s,c} \neq t_s$ (and $t_{s,c} \neq t_s + 1$ during the transition from t_s to $t_s + 1$) as in Figure 5, the server does not hold $sk_s^{t_{s,c}}$ and cannot compute the early handshake-traffic, the server/client handshake traffic or the early traffic secrets (denoted EHTS′, SHTS′, CHTS′, and ETS′), and therefore cannot recover K_s, the client certificate or any potential early application data. The server thus rejects the first four stages.
The main idea in this case is close to that of a `HelloRetryRequest` in TLS 1.3 [43]. The server's response to the client does not contain a KEM_c ciphertext, indicating that their time periods did not match, but however contains an ephemeral KEM ciphertext. The client can then decapsulate the ciphertext, recover an ephemeral secret, and restart as in the case of matching time periods; the now-established ephemeral secret essentially takes the place of the semi-static one. The protocol is thereby delayed by a single round trip. The details are given in Figure 5 in the appendix.

Discussion. In case an adversary were to change $t_{s,c}$, the parties would not execute the correct branch of the protocol. However, the handshake secrets they would compute would simply differ without their secrecy being threatened, and the confirmation messages would not pass verification, i.e., both parties would be aware that the `ClientHello` message was tampered with.

Note also that it is important for the client to only use authentic semi-static keys (i.e., obtained from a previous handshake in which the server was explicitly authenticated), as otherwise an adversary could send the client a semi-static key it generated itself and later recover the client's identity in a past handshake by only corrupting the server long-term key.

Lastly, exporter and resumption secrets could also be derived from the master secret as in TLS 1.3. Though omitted from the protocol description, they can be readily added to the key schedule and would satisfy the same security properties as the application transport secrets.

Application to KEMTLS with Client Authentication. The idea of introducing semi-static keys to shorten the handshakes by a full round trip (while maintaining forward identity protection) can also be applied to the KEMTLS

[4] The server does so once per client; the client will then switch to the next key for subsequent handshakes.

CAHTS/SAHTS	Client/Server Auth. Handshake Traffic Secret
CATS/SATS	Client/Server Application Transport Secret
CHTS/SHTS	Client/Server Handshake-Traffic Secret
dES/dHS/dIMS	Derived Early/Handshake/Intermediate Master Secret
ES/HS/(I)MS	Early/Handshake/(Intermediate) Master Secret
fk_c/fk_s	Client/Server Finished Key

Table 1: Glossary of values derived in the protocol.

protocol (without pre-loaded certificates) with client authentication [46, Appendix C.1]: it suffices to run the full protocol for the first handshake in a time period and have the server send the current semi-static public key along with the server certificate. The client can then save the long-term and semi-static public keys and for subsequent handshakes in the time period run the protocol from this section.

4 Security Model

This section introduces the model to capture security of the key-exchange protocol presented in Section 3. It is close to the model for authenticated key exchange proposed by Dowling, Fischlin, Günther and Stebila [18, 19] and that for KEMTLS by Schwabe, Stebila, and Wiggers [46]. Their models follow a line of work [24, 28] concerned with *multi-stage* key exchange protocols in which keys are computed at multiple stages of each single protocol execution. It originated from Bellare and Rogaway's model [6] that introduced the paradigm of session-key indistinguishability, and for which Brzuska et al. [10, 11] formalized the composability with arbitrary protocols based on symmetrically distributed keys.

In the security model, the adversary controls the network and can passively eavesdrop, modify and orchestrate the communication across several concurrent sessions of the protocol. The adversary can further expose long-term and semi-static secrets of honest parties as well as the keys established during protocol runs (individually per stage). The protocol is then deemed *multi-stage secure* if such an adversary cannot distinguish a key established at a stage of a non-compromised ("*fresh*") session from a uniformly random key.

Authentication. The model supports *mutual* authentication, as required in the scenario of IoT or embedded devices. For the authentication of each stage key, *implicit* and *explicit* authentication are distinguished. Implicit authentication refers to the property that the stage key can only be recovered by the intended partner, whereas explicit authentication guarantees that the partner actively participated in the protocol and also established a stage key. The authentication of a stage key can further be lifted from unauthenticated or implicit to explicit once a later stage of the protocol is accepted: a stage key can be *retroactively* explicitly authenticated.

Forward Secrecy. The model further covers forward secrecy, the notion that stage keys remain secret even if the long-term keys involved in its computation are later compromised. As the protocol in Section 3 introduces server semi-static keys (i.e., keys that are periodically refreshed) for servers in addition to long-term keys, the notion of forward secrecy is here refined to also take the compromise of such keys into account.

More precisely, the model considers two types of forward secrecy determined by whether the semi-static key used to compute a stage key may be corrupted.

- A stage key satisfies *(full) forward secrecy* if the adversary remained passive until the session accepted the stage or did not corrupt the long-term key of the intended communication partner before the latter was explicitly authenticated. The semi-static key used to compute the stage key may be corrupted at any time.
- A stage key satisfies *delayed forward secrecy* if, in addition to the previous conditions, the adversary did not corrupt the semi-static key used to compute the stage key. In particular, if the long-term key of the intended partner is not corrupted before the semi-static key expires, the secrecy of the stage key is equivalent to that of a key satisfying full forward secrecy. This (informal) definition of delayed forward secrecy is in this sense related to Boyd and Gellert's [9].

Key Usage. The use of stage keys is also specified, i.e., whether a key is meant to be used internally within the protocol (e.g., to encrypt handshake traffic) or externally (for example to protect application messages).

Replays. The model further captures that the initial, first-flight keys are *replayable*: an attacker may copy the client's initial messages and send them to the server (again), leading to multiple server sessions sharing the same keys with that one original client. This is due to the key being derived without interaction (in zero round-trip time) and hence with no active contribution from the server side. Following [19, 25], the model distinguishes between replayable and non-replayable stages, catering for this situation (which would otherwise lead to a violation of partnering uniqueness), while still demanding that keys remain indistinguishable from random, even when replayable.

4.1 Syntax

Similarly to the model for TLS 1.3 proposed by Dowling et al. [18, 19], a multi-stage key-exchange protocol is characterized by a set of values. In the present case, these are as follows.

- $M \in \mathbb{N}$: the number of protocol stages, i.e., the number of keys derived in a session.
- $FS \in \{dfs, fs\}^M$: for $i \in [\![M]\!]$, FS_i specifies the type of forward secrecy expected from the key computed at stage i.

- $iauth_s \in \{1, \ldots, M\}$: a variable indicating the stage from which the server is implicitly authenticated.
- $iauth_c \in \{iauth_s, \ldots, M\}$: a variable that specifies the stage from which the client is implicitly authenticated. It is here assumed that the server is always the first party to be authenticated, which is in line with identity protection.
- $eauth = ((u_1, m_1), \ldots, (u_1, m_M))$ with $u_i \leq m_i \in \{i, \ldots, M, \infty\}$ for all $i \in [\![M]\!]$: a pair tuple encoding the intended explicit-authentication pattern. For example, the i-th pair (u_i, m_i) means that the key computed at stage i reaches explicit server authentication once stage $u_i \geq i$ is accepted and explicit client and server (i.e., mutual) authentication once stage m_i is accepted[5]. The value ∞ indicates that unilateral ($u_i = \infty$) or mutual ($m_i = \infty$) authentication is never reached for the corresponding stage.
- $use \in \{internal, external\}^M$: the usage indicator for each stage key, with use_i denoting the usage of the key established at stage i. An internal key may be used in the key exchange protocol (and also externally), whereas an external key is not to be used in the protocol in order to allow for generic composition.
- $replay \in \{replayable, nonreplayable\}^M$: the replayability indicator for each stage, with $replay_i$ indicating whether the key established at stage i can be replayed. A replayable key may be derived non-uniquely in more than one responder session.

The set of participants is denoted $ID = \Gamma \cup \Sigma$, with Γ the set of clients and Σ the set of servers (the union need not be disjoint). Each identity $id \in ID$ is associated with a certified long-term public key pk_{id} and a corresponding secret key sk_{id}. The root certificate of the certificate authority is assumed to be pre-distributed to all parties. The clients are also assumed to be given the certificates of the servers to which they may connect, *prior to any handshake request.* In addition to long-term keys, each server participant $s \in \Sigma$ also holds a pair of semi-static keys $\left(pk_s^{t_s}, sk_s^{t_s}\right)$ for each local time period $t_s \in \mathbb{N}$ (the pair is generated shortly before the beginning of period t_s), and these are erased when the next period begins. The semi-static public keys are *not* assumed to be given to the clients out of band; the servers must transmit them during protocol executions.

Each participant (client or server) can run several concurrent instances of the protocol, with each local instance referred to as a *session*, and each session consisting of multiple *stages*, i.e., computation steps at which keys are derived. A session is identified with a pair $\sigma = (id, n) \in ID \times \mathbb{N}$ which denotes the n-th local session of participant id. A participant running a session locally maintains the session-specific information below.

- id : the identity of the session owner.
- $pid \in ID \cup \{*\}$: the identity of the intended partner. In case[6] $id \in \Sigma$ and the identity of the intended partner client is currently unknown, the special

[6] Post-specified peers are only considered for server sessions as client sessions must be given from the beginning of the execution the server identity for which they use the semi-static public key.

symbol $*$ is used, and it may later be updated to a specific identity once by the protocol. That is to say, the identity of the communication partner may only be known during the execution of the protocol (e.g., through exchanged certificates), capturing post-specified peers [13].

- $role \in \{initiator, responder\}$: the role of the session.
- $status \in \{\perp, running, accepted, rejected\}^M$: the status of each stage. It is initially set to $(running, \perp, \ldots, \perp)$. $status_i \leftarrow accepted$ once the stage i is accepted. $status_i \leftarrow rejected$ if the stage i is rejected.
- $stage \in \{0, \ldots, M\}$: indicates the last completed stage. This variable is initially set to 0, and $stage \leftarrow i$ if $status_i$ is updated to *accepted* or *rejected*.
- $cid \in (\{0,1\}^* \cup \{\perp\})^M$: records the contributive identifier at each stage. For all $i \in [\![M]\!]$, cid_i is initially set to $\perp$ and may be updated until the stage is either accepted or rejected.
- $sid \in (\{0,1\}^* \cup \{\perp\})^M$: holds the session identifier at each stage. For all $i \in [\![M]\!]$, sid_i is initially set to $\perp$ and is updated (only) once upon acceptance of stage i.
- $key \in (\{0,1\}^* \cup \{\perp\})^M$: holds the key established at each stage. For all $i \in [\![M]\!]$, key_i is initially set to $\perp$ and is set upon acceptance of stage i.

A variable, e.g., id, pertaining to a particular session σ is denoted $\sigma.id$.

Partnering. Two sessions σ and σ' are considered partnered at stage i if they are distinct and share the same session identifier at that stage, i.e., $\sigma \neq \sigma'$ and $\sigma.sid_i = \sigma'.sid_i \neq \perp$.

4.2 Security Definitions

As in the work of Brzuska et al. [11], the security of key-exchange protocols is here defined via two notions: *multi-stage security* (which here combines session-key indistinguishability and explicit authentication) and *match security*. Session-key indistinguishability ensures that keys computed at non-compromised stages are indistinguishable from uniformly random keys. Explicit authentication captures the idea that no session stage expecting an explicitly authenticated peer maliciously accepts without such. Match security finally guarantees that in a multi-stage setting, session identifiers correctly identify sessions that are partnered in effect. These two notions are formalized via respective security games in which an adversary interacts with a challenger. The adversary is given access to challenger-controlled oracles through which it can initiate concurrent protocol executions, interact with honest participants (run by the challenger), and control the messages they send, i.e., forward, delay, modify, drop or even change the order of the messages. At the end of the interaction, the challenger returns a bit indicating whether conditions that contradict the security notion being defined are fulfilled. The advantage of the adversary in the game is then defined as the probability of this event.

Game Variables. Throughout the interaction with the adversary, the challenger maintains the following variables for each server identity $s \in \Sigma$.

- $t_s \in \mathbb{N}$: the current time period of server s, initialized to 0.
- (pk'_s, sk'_s) : the pair of keys for the current time period, initialized during the game setup. The value of sk'_s while variable t_s holds a value $t \in \mathbb{N}$ is denoted sk^t_s in Definition 4.2.
- $\left(pk''_s, sk''_s\right)$: the pair of keys for the next time period, initialized to $(\perp, \perp)$.

For each client identity$c \in \Gamma$, the challenger maintains

- a pair $(t_{s,c}, pk'_{s,c})$ of time period and semi-static key for each server identity $s \in \Sigma$. Each such pair is initialized to $(-\infty, \perp)$ and may be updated throughout the game.

The challenger additionally maintains the following game variables for each session σ that it runs.

- $period \in \mathbb{N} \cup \{-\infty\}$: records the time period of the server semi-static key used by the session. If $\sigma.role = initiator$, then $period \leftarrow t_{\sigma.pid,\sigma.id}$ and cannot be updated. If $\sigma.role = responder$, then $period \leftarrow t_{\sigma.id}$ and may be updated once to $period + 1$ once by the protocol (during the transition from a period to another).
- $revealed \in \{\text{TRUE}, \text{FALSE}\}^M$: a vector recording the stage keys that were revealed to the adversary. All entries are initially set to FALSE.
- $tested \in \{\text{TRUE}, \text{FALSE}\}^M$: a vector recording which stage keys were tested by the adversary, i.e., for which the adversary was returned either the stage key or a random key. All entries are initially set to FALSE.

Game Oracles. The adversary is given access to the following oracles in the security games.

- $\mathsf{NewSession}(p, q, role)$: creates a new session σ with owner $\sigma.id \leftarrow p$, intended peer $\sigma.pid \leftarrow q$ and role $\sigma.role \leftarrow role$. If $role = responder$, the identity q may be unspecified, i.e., set to $*$. The oracle returns σ.
- $\mathsf{Send}(\sigma, m)$: returns $\perp$ if session σ does not exist (i.e., was not started by a query to oracle $\mathsf{NewSession}$), otherwise runs the protocol algorithm of the participant with the current state on input m, updates the state and returns the response (if any) and $(\sigma.stage, \sigma.status_{\sigma.stage})$. To initiate a session in case $\sigma.role = initiator$, the adversary may submit the special message $m := init$. If $\sigma.status_{\sigma.stage}$ is updated to $accepted$, then the execution is halted. This allows the adversary to test an internal key before it is used in the protocol, which is forbidden at a later point in the execution to prevent trivial wins (see oracle Test below). Once the execution is halted, the adversary may perform operations on other sessions, test the stage key (i.e., submit a $\mathsf{Test}(\sigma, \sigma.stage)$ query) or submit a $\mathsf{Send}(\sigma, continue)$ query to resume the computation.
  - If there exists a distinct session $\sigma' \neq \sigma$ such that $\sigma.sid_{\sigma.stage} = \sigma'.sid_{\sigma.stage}$ and $\sigma'.tested_{\sigma.stage} = \text{TRUE}$, then

▷ the challenger sets $\sigma.tested_{\sigma.stage} \leftarrow$ TRUE. This ensures that if a partnered session was already tested, then subsequent test queries are appropriately answered.

▷ if $use_{\sigma.stage} = internal$, then $\sigma.key_{\sigma.stage} \leftarrow \sigma'.key_{\sigma.stage}$. That is, if the key is internal, then it is set consistently with the key from the partnered session. (This assumes the property that if two partnered sessions accept a stage key, then these are equal, i.e., a guarantee of match security to be formally defined later.)

∗ If the adversary resumes the computation and $\sigma.stage < M$, then the challenger sets $\sigma.status_{\sigma.stage+1} \leftarrow running$ and the oracle returns the next protocol message and $(\sigma.stage + 1, \sigma.status_{\sigma.stage+1})$.

– $\mathsf{Reveal}(\sigma, i)$: returns $\perp$ if session σ does not exist or if $\sigma.status_i \neq accepted$, otherwise returns $\sigma.key_i$. The challenger then sets $\sigma.revealed_i \leftarrow$ TRUE.

– $\mathsf{NextPeriod}(s \in \Sigma)$: if $\left(pk''_s, sk''_s\right) \neq (\perp, \perp)$, i.e., if server s is transitioning from t_s to $t_s + 1$, then the oracle returns $\perp$, otherwise the oracle overwrites $\left(pk''_s, sk''_s\right)$ with a fresh pair of keys and returns pk''_s.

– $\mathsf{EndCurrentPeriod}(s \in \Sigma)$: sets $t_s \leftarrow t_s + 1$, $\left(pk'_s, sk'_s\right) \leftarrow \left(pk''_s, sk''_s\right)$ and $\left(pk''_s, sk''_s\right) \leftarrow (\perp, \perp)$.

– $\mathsf{Corrupt}(id)$: returns the long-term secret key sk_{id} of participant id.

– $\mathsf{SemiStaticCorrupt}(s \in \Sigma, d \in \{0, 1\})$: returns sk'_s if $d = 0$; the key in period t_s is now considered corrupt. If $d = 1$, returns sk''_s; if $sk''_s \neq \perp$, the key in period $t_s + 1$ is now considered corrupt.

– $\mathsf{Test}(\sigma, i)$: based on a bit b fixed throughout the game, this oracle returns either the key that σ computed at stage i if $b = 0$ or a uniformly random key if $b = 1$, unless σ does not exist or conditions which prevent the adversary from readily distinguishing the two cases are not met.

∗ If σ does not exist, then return $\perp$.

∗ If $\sigma.status_i \neq accepted$ or there exists a session σ' (not necessarily distinct) such that $\sigma.sid_i = \sigma'.sid_i$ and $\sigma'.tested_i =$ TRUE, then the oracle returns $\perp$. In other words, a stage key is tested at most once.

∗ If $\sigma.use_i = internal$ and there exists a session σ' (not necessarily distinct) such that $\sigma.sid_i = \sigma'.sid_i$ and $\sigma'.status_{i+1} \neq \perp$, then the oracle returns $\perp$. This guarantees that a potential partnered session, which may have already established this internal key, has not already used it.

∗ The challenger sets $\sigma.tested_i \leftarrow$ TRUE.

∗ $K \leftarrow \sigma.key_i$ if $b = 0$, otherwise K is drawn uniformly at random from the key space.

∗ If $\sigma.use_i = internal$, the challenger sets $\sigma.key_i \leftarrow K$ to remain consistent with later use of the key.

∗ For any session $\sigma' \neq \sigma$ such that $\sigma.sid_i = \sigma'.sid_i$ and $\sigma'.status_i \leftarrow accepted$, the challenger sets $\sigma'.tested_i \leftarrow$ TRUE, and if additionally $\sigma.use_i = internal$, then the challenger sets $\sigma'.key_i \leftarrow K$.

∗ The oracle ultimately returns K to the adversary.

Remark. Although the adversary can reveal session keys and corrupt long-term and semi-static keys, the model does not go as far as capturing leakage of variables internal to a session as in other models [12,37]. In particular, the adversary is not given access to any potential ephemeral values. These are assumed to be in practice erased as soon as they are no longer needed, which is crucial to allow for forward secrecy.

Match Security. A key-exchange protocol satisfies the match property if session identifiers properly determine which sessions are effectively partnered, in the sense expressed by the winning conditions of the following security game.

Definition 4.1 (Match Security). *Let* KE *be a multi-stage key-exchange protocol characterized by a tuple* $(M, FS, iauth_s, iauth_c, eauth, use, replay)$*, and ID be a set of participants. Consider an adversary* $\mathscr{A}$ *which interacts with the challenger of the game defined below and further denoted* $G^{\mathrm{Match}}_{\mathsf{KE},\mathscr{A}}$*.*

Setup. *The challenger generates a pair of long-term keys* (pk_{id}, sk_{id}) *for each participant identity* $id \in ID$*. For each server identity* $s \in \Sigma$*, the challenger generates a fresh pair of semi-static keys* $\left(pk'_s, sk'_s\right)$ *(for the initial time period). The challenger samples a uniformly random bit* $b \leftarrow_{\$} \{0,1\}$ *for the test oracle.*

Query. $\mathscr{A}$ *is given access to all the oracles specified above (b parametrizes the test oracle).*

Stop. $\mathscr{A}$ *ultimately terminates its computation with no output.*

$G^{\mathrm{Match}}_{\mathsf{KE},\mathscr{A}}$ *returns* 1 *(i.e.,* $\mathscr{A}$ *wins the game) if at least one of the following conditions holds.*

1. *More than two sessions share the same identifier at some non-replayable stage, i.e., three pairwise distinct sessions* σ*,* σ' *and* σ'' *and a stage* $i \in [\![M]\!]$ *such that* $\sigma.sid_i = \sigma'.sid_i = \sigma''.sid_i \neq \perp$ *and* $replay_i = nonreplayable$*.*
2. *Two sessions share the same identifier at some stage but have non-opposite roles (except for potential multiple responders in replayable stages), i.e., two distinct sessions* σ *and* σ' *and a stage* $i \in [\![M]\!]$ *such that* $\sigma.sid_i = \sigma'.sid \neq \perp$*, and* $\sigma.role = \sigma'.role$ *and* $replay_i = nonreplayable$*, or* $\sigma.role = \sigma'.role = initiator$*.*
3. *Two sessions share the same identifier at some stage but computed different stage keys,i.e., two sessions* σ *and* σ' *and a stage* $i \in [\![M]\!]$ *such that* $\sigma.sid_i = \sigma'.sid_i \neq \perp$ *and* $\sigma.key_i \neq \sigma'.key_i$*.*
4. *Two sessions share the same identifier but have distinct or unspecified contributive identifiers at some stage, i.e., two distinct sessions* σ *and* σ' *and a stage* $i \in [\![M]\!]$ *such that* $\sigma.sid_i = \sigma'.sid_i \neq \perp$ *and either* $\sigma.cid_i \neq \sigma'.cid_i$ *or* $\sigma.cid_i = \sigma'.cid_i = \perp$*.*
5. *Two distinct stages share the same session identifier, i.e., two sessions* σ *and* σ' *(not necessarily distinct) and two stages* $i \neq j \in [\![M]\!]$ *such that* $\sigma.sid_i = \sigma'.sid_j \neq \perp$*.*

6. *A stage has (retroactively) reached explicit authentication, but the partnered session does not belong to the intended peer. Formally, there exist two distinct sessions σ and σ' with $\sigma.role = initiator$ and $\sigma'.role = responder$ as well as two stages $j \leq i \in [\![M]\!]$ such that $\sigma.sid_i = \sigma'.sid_i \neq \perp$ and $\sigma.sid_j = \sigma'.sid_j \neq \perp$, and either*
   - *$\sigma.eauth_{j,1} \leq i$ and $\sigma.pid \neq \sigma'.id$, or*
   - *$\sigma'.eauth_{j,2} \leq i$ and $\sigma'.pid \neq \sigma.id$.*

Protocol KE *satisfies $(T, q_{\mathcal{O}}, \varepsilon)$-match security for*

$$\mathcal{O} \in \{\mathsf{NewSession}, \mathsf{Send}, \mathsf{Reveal}, \mathsf{NextPeriod}, \mathsf{EndCurrentPeriod}, \mathsf{Corrupt}, \mathsf{SemiStaticCorrupt}, \mathsf{Test}\}$$

if for any adversary $\mathscr{A}$ that runs in time at most $T(\lambda)$ and makes at most $q_{\mathcal{O}}(\lambda)$ queries, the advantage $\Pr\left[G^{\mathrm{Match}}_{\mathsf{KE},\mathscr{A}} \to 1\right]$ of $\mathscr{A}$ in the game is at most $\varepsilon(\lambda)$.

Multi-Stage Security. The notion of multi-stage security captures the idea that keys computed at non-compromised stages should be indistinguishable from uniformly random keys, and it includes aspects such as forward secrecy and implicit authentication. Non-compromised stages are formally defined through the sub-notion of freshness. Multi-stage security also covers explicit authentication, which is formalized via the sub-notion of malicious acceptance. The latter requires a partnered session to exist once a stage has (retroactively) reached explicit authentication, provided that the adversary did not corrupt the secret keys of the partner up to that computation step.

Definition 4.2 (Freshness). *The i-th stage of a session σ is considered* fresh *if all of the conditions hereafter hold.*

1. *The key session σ or a potential partner computed at stage i was not revealed, i.e., for any session σ' such that $\sigma.sid_i = \sigma'.sid_i$, $\sigma'.revealed_i = \mathrm{FALSE}$.*
2. *If $\sigma.FS_i = fs$, then*
   (a) *(Honest contributive partner) there exists a session $\sigma' \neq \sigma$ such that $\sigma.cid_i = \sigma'.cid_i$ and $\sigma.role \neq \sigma'.role$, **or***
   (b) *(Implicit authentication) the partner is implicitly authenticated from a stage at most i ($\sigma.iauth_s \leq i$ if $\sigma.role = initiator$ and $\sigma.iauth_c \leq i$ otherwise) and $sk_{\sigma.pid}$ is not corrupt, **or***
   (c) *(Forward secrecy) stage i reached explicit partner authentication (that is, $\sigma.status_{\sigma.eauth_{i,1}} = accepted$ if $\sigma.role = initiator$ and $\sigma.status_{\sigma.eauth_{i,2}} = accepted$ otherwise) and the adversary did not corrupt $sk_{\sigma.pid}$ before σ accepted the corresponding stage.*
3. *If $\sigma.FS_i = dfs$, then*
   (a) *(Server implicit authentication) $\sigma.role = initiator$ and either $sk^{\sigma.period}_{\sigma.pid}$ is not corrupt or the server is implicitly authenticated from a stage at most i ($\sigma.iauth_s \leq i$) and $sk_{\sigma.pid}$ is not corrupt, **or***
   (b) *$\sigma.role = responder$ and*

i. (Honest contributive initiator) there exists a session $\sigma' \neq \sigma$ *such that* $\sigma.cid_i = \sigma'.cid_i$ *and* $\sigma.role \neq \sigma'.role$*, and* $sk_{\sigma.id}^{\sigma.period}$ *is not corrupt,* ***or***

ii. (Client implicit authentication) the client is implicitly authenticated from a stage at most i *(*$\sigma.iauth_c \leq i$*) and* $sk_{\sigma.pid}$ *is not corrupt,* ***or***

iii. (Delayed forward secrecy) stage i *reached explicit mutual authentication (*$\sigma.status_{\sigma.eauth_{i,2}} = accepted$*),* $sk_{\sigma.pid}$ *was not corrupted before* σ *accepted stage* $eauth_{i,2}$ *and* $sk_{\sigma.id}^{\sigma.period}$ *is not corrupt.*

On Freshness. Beyond ruling out the trivial attack of both revealing and testing a session key (within the same or a partnered session[7]), the above definition distinguishes two main cases depending on the expected forward secrecy of the considered stage key.

In the case of (full) forward secrecy (i.e., $\sigma.FS_i = fs$), the rationale behind the freshness conditions is that ephemeral values are used in the computation of these stage keys and are erased once the session is terminated. The conditions then exclude situations in which the adversary has access to both the ephemeral values and the long-term key of the intended partner and could thus reconstruct the stage key. For instance, if the adversary did not corrupt the long-term key of the partner before the latter was explicitly authenticated, it is guaranteed that the honest intended partner participated in the protocol and the ephemeral values are therefore unknown to the adversary.

For stage keys that rather satisfy delayed forward secrecy (i.e., $\sigma.FS_i = dfs$), no fully ephemeral values are involved in their computation. The closest equivalent in this case are semi-static values that are generated by the client and are recoverable with the secret semi-static keys. However, since servers have semi-static keys but clients do not, there is an asymmetry in the conditions on client sessions and those on server sessions. As for clients, once a client session is closed and the internal values are erased, the semi-static KEM encapsulation can only be recovered with the semi-static secret key. This means that the adversary cannot distinguish from random such a client stage key if it does not corrupt the semi-static secret key used to compute it; the servers are in a sense also authenticated through their semi-static keys. Note that the freshness conditions do not involve the stage from which the server is explicitly authenticated as there is no fresh semi-static contribution from the server (because clients do not have semi-static keys). The guarantee that the honest intended partner server was live is therefore irrelevant. For server keys, the conditions are closer to those of full forward secrecy, except that when the corruption of the partner client key is allowed, the corruption of the semi-static key is not. In case there is an honest contributive identifier, the adversary could otherwise compute all stage keys by also corrupting the long-term keys of both parties. In case the client is explicitly

[7] Note that in replayable stages more than two sessions maybe be partnered due potential non-unique responder sessions. Multi-stage key indistinguishability still requires that as long as the key in *none* of these partnered sessions is trivially revealed, it remains indistinguishable in *all* of these partnered sessions.

authenticated and her long-term key was not corrupted before that, it is indeed guaranteed that there is a fresh honest semi-static contribution from the client, but corrupting of the corresponding semi-static key would allow the adversary to recover it.

Definition 4.3 (Malicious Acceptance). *A session σ is said to have maliciously accepted a stage i if the latter (retroactively) reached explicit partner authentication (i.e., stage $\sigma.eauth_{i,1}$ if $\sigma.role = initiator$ and stage $\sigma.eauth_{i,2}$ otherwise), the adversary did not corrupt $sk_{\sigma.pid}$ before σ accepted the corresponding stage and there is no session $\sigma' \neq \sigma$ such that $\sigma.sid_i = \sigma'.sid_i$.*

Definition 4.4 (Multi-Stage Security). *Let* KE *be a multi-stage key-exchange protocol characterized by a tuple $(M, FS, iauth_s, iauth_c, eauth, use, replay)$, and ID a set of participants. Consider an adversary $\mathscr{A}$ interacting with the challenger of the game defined below and further denoted $G^{\text{Multi-Stage}}_{\mathsf{KE},\mathscr{A}}$.*

Setup. *The challenger generates a pair of long-term keys (pk_{id}, sk_{id}) for each participant identity $id \in ID$. For each server identity $s \in \Sigma$, the challenger generates a fresh pair of semi-static keys (pk'_s, sk'_s) (for the initial time period). The challenger samples a uniformly random bit $b \leftarrow_{\$} \{0,1\}$ for the test oracle.*

Query. *$\mathscr{A}$ is given access to all the oracles defined above (b parametrizes the test oracle).*

Stop. *$\mathscr{A}$ ultimately terminates its computation and returns a bit b'.*

Finalize. *$-$ $b' \leftarrow_{\$} \{0,1\}$ if a tested stage is not fresh.*
$-$ $b' \leftarrow b$ if a session maliciously accepts a stage.
$G^{\text{Multi-Stage}}_{\mathsf{KE},\mathscr{A}}$ returns 1 if $b = b'$ and otherwise 0.

Protocol KE *satisfies $(T, q_{\mathscr{O}}, \varepsilon)$-multi-stage security for*

$$\mathscr{O} \in \{\mathsf{NewSession}, \mathsf{Send}, \mathsf{Reveal}, \mathsf{NextPeriod}, \mathsf{EndCurrentPeriod}, \mathsf{Corrupt}, \mathsf{SemiStaticCorrupt}, \mathsf{Test}\}$$

if for any adversary $\mathscr{A}$ that runs in time at most $T(\lambda)$ and makes at most $q_{\mathscr{O}}(\lambda)$ queries, the advantage $\left|\Pr\left[G^{\text{Multi-Stage}}_{\mathsf{KE},\mathscr{A}} \to 1\right] - 1/2\right|$ of $\mathscr{A}$ in the game is at most $\varepsilon(\lambda)$.

Comparison with Existing Models. A major difference from existing models is the introduction of time periods. Semi-static keys have been introduced in the original multi-stage model to analyze the QUIC protocol [24] and have also been used to study the later-deprecated Diffie–Hellman-based 0-RTT mode in early TLS 1.3 drafts [25] based on the OPTLS design [35]. The model handles these via the time-related identity and session game variables, the oracles that give the adversary full control over time periods as well as the oracle to corrupt semi-static keys, and a refinement of forward secrecy that takes semi-static keys into account. These considerations naturally impact the freshness predicate, which is crucial to the formal definition of key indistinguishability.

Apart from these aspects, the present model is closest to the model for TLS 1.3 due to Dowling et al. [18,19] but it still departs from it in several ways. First, it covers explicit authentication through the notion of malicious acceptance (inspired by the definition in the KEMTLS model [46]), in contrast to the model from [18,19] which only formalizes implicit authentication. The model also considers a single authentication mode (i.e., mutual) per protocol instead of several ones as in the TLS 1.3 model [18,19]. This simplification is possible because both parties always authenticate themselves in the protocol from Section 3. Besides, the model in [18,19] also deals with cases in which parties share symmetric keys obtained from previous sessions (i.e., so-called pre-shared keys) and resumption mechanisms, but these are not considered in this paper. Similarly to the replayable 0-RTT keys derived in TLS 1.3 pre-shared key handshake, the model captures the replayability of the initial keys in the protocol from Section 3 that are derived without active server contribution.

Another major technical difference from the model by Dowling et al. [18,19] is in the definition of the freshness predicate. Indeed, Definition 4.2 involves the stages from which the parties are explicitly authenticated whereas key indistinguishability in their model is only concerned with implicit authentication. They can do so because implicit and explicit authentication happen simultaneously in the TLS 1.3 protocol. It thereby naturally excludes the attack in which an adversary impersonates the intended partner up to the stage of implicit authentication, halts the protocol execution before reaching explicit authentication and later corrupts the long-term key of the intended partner. However, this attack is clearly possible in the protocol from Section 3, and that is why the conditions of forward secrecy enforce that if the adversary ever corrupts the long-term key of the intended partner, then it must be after the intended partner is explicitly authenticated, which ensures that the adversary does not have access to the ephemeral values of the execution. This concern also arises in the analysis of KEMTLS protocol [46] which introduces several levels of forward secrecy that tacitly integrate different levels of authentication. Although forward secrecy and authentication are related, the above model syntactically disentangles the two properties for clarity and remains in this sense closer to the one in [18,19].

5 Security Analysis

This section specifies the syntactic values of the Section 3 protocol and then discusses the security properties that it satisfies in the model from Section 4.

5.1 Properties

The characteristic values of the protocol and the session-specific values are here defined, in both the cases of matching and unmatching time periods.

Matching Time Periods. In this case, the properties the protocol aims to satisfy are as follows.

- $M = 8$: the protocol has eight stages as shown in Figure 4.
- $FS = (dfs, dfs, dfs, dfs, fs, fs, fs, fs)$: the first four keys satisfies delayed forward secrecy, the others full forward secrecy.
- $iauth_s = 2$, $iauth_c = 5$: the server is implicitly authenticated from stage 2 on, the client from stage 5 on.
- $eauth = ((7,8),(7,8),(7,8),(7,8),(7,8),(7,8),(7,8),(8,8))$: The server is explicitly authenticated from stage 7 on, the client from stage 8 on.
- $use = (internal : \{1,2,3,5,6\}, external : \{4,7,8\})$: The keys derived at stages 1–3 and 5–6 are used to encrypt handshake traffic, i.e., for internal use.
- $replay = (replayable : \{1,2,3,4\}, nonreplayable : \{5,6,7,8\})$: The first four stage keys, without active server contribution, are replayable; all other keys are not.

Session and Contributive Identifiers. Recall that each instance of a protocol algorithm maintains a set of session identifiers and contributive identifiers. The session identifier at each stage is computed once the stage is accepted, and it consists of all the handshake messages up to the acceptance of the stage (excluding the final finished messages). It the present case, denoting by $\texttt{SC} \coloneqq \texttt{ServerCertificate}$ the server certificate $cert[pk_s]$, the session identifiers are

$$
\begin{aligned}
sid_1 &= (\text{“EHTS”}, \texttt{SC}, \texttt{CH}, \texttt{SSKC}),\\
sid_2 &= (\text{“SHTS”}, \texttt{SC}, \texttt{CH}, \texttt{SSKC}, \texttt{CKC}),\\
sid_3 &= (\text{“CHTS”}, \texttt{SC}, \texttt{CH}, \texttt{SSKC}, \texttt{CKC}),\\
sid_4 &= (\text{“ETS”}, \texttt{SC}, \texttt{CH}, \texttt{SSKC}, \texttt{CKC}),\\
sid_5 &= (\text{“SAHTS”}, \texttt{SC}, \texttt{CH}, \texttt{SSKC}, \texttt{CKC}, \texttt{CC}, \texttt{SH}, \texttt{SKC}),\\
sid_6 &= (\text{“CAHTS”}, \texttt{SC}, \texttt{CH}, \texttt{SSKC}, \texttt{CKC}, \texttt{CC}, \texttt{SH}, \texttt{SKC}),\\
sid_7 &= (\text{“SATS”}, \texttt{SC}, \texttt{CH}, \texttt{SSKC}, \texttt{CKC}, \texttt{CC}, \texttt{SH}, \texttt{SKC}, \texttt{SPK}^*, \texttt{EE}),\\
sid_8 &= (\text{“CATS”}, \texttt{SC}, \texttt{CH}, \texttt{SSKC}, \texttt{CKC}, \texttt{CC}, \texttt{SH}, \texttt{SKC}, \texttt{SPK}^*, \texttt{EE}, \texttt{SF}).
\end{aligned}
$$

The contributive identifiers should capture when a session is fresh due to an honest contribution by a party. Here, this honest contribution refers to the ephemeral key share K_e, the relevant messages are CH and SH. The parties hence compute the contributive identifiers as follows.

- Upon sending (resp. receiving) the SemiStaticKEMCiphertext message, the client (resp. server) sets $cid_1 \leftarrow sid_1$.
- Upon sending (resp. receiving) the ClientKEMCiphertext message, the client (resp. server) sets $cid_2 \leftarrow sid_2$, $cid_3 \leftarrow sid_3$, and $cid_4 \leftarrow sid_4$.
- Upon sending (resp. receiving) the ClientCertificate message, the client (resp. server) sets $cid_5 \leftarrow$ (“SAHTS”, SC, CH, SSKC, CKC, CC), $cid_6 \leftarrow$ (“CAHTS”, SC, CH, SSKC, CKC, CC), and $cid_7 \leftarrow$ (“SATS”, SC, CH, SSKC, CKC, CC).
- After computing stages 5, 6 and 7, both the client and the server update $cid_5 \leftarrow$ (“SAHTS”, SC, CH, SSKC, CKC, CC, SH), $cid_6 \leftarrow$ (“CAHTS”, SC, CH, SSKC, CKC, CC, SH), and $cid_7 \leftarrow$ (“SATS”, SC, CH, SSKC, CKC, CC, SH).
- Client and server set $cid_8 \leftarrow$ (“CATS”, SC, CH, CC, SH) after they compute stage 8.

Unmatching Time Periods. In case the time periods of the client and of the server do not match, the protocol targets the following properties.

- $M = 12$ stages as in Figure 5.
- $FS = (\mathit{dfs}, \mathit{dfs}, \mathit{dfs}, \mathit{dfs}, \mathit{fs}, \mathit{fs}, \mathit{fs}, \mathit{fs}, \mathit{fs}, \mathit{fs}, \mathit{fs}, \mathit{fs})$: the first four stage keys (stages 1'–4'), only accepted by the client, satisfy delayed forward secrecy and all the others full forward secrecy.
- $\mathit{iauth}_s = 2$, $\mathit{iauth}_c = 5$: the server is implicitly authenticated from stage 2 on[8], the client from stage 5 on.
- $\mathit{eauth} = (\infty, \infty), (\infty, \infty), (\infty, \infty), (\infty, \infty), (7, 8), (7, 8), (7, 8), (7, 8), (7, 8), (7, 8), (7, 8), (8, 8)$: the server is explicit authenticated from stage 7 on, the client from stage 8 on. The rejected initial four keys are never explicitly authenticated.
- $\mathit{use} = (\mathit{internal} : \{1', 2', 3', 1, 2, 3, 5, 6\}, \mathit{external} : \{4', 4, 7, 8\})$: the keys computed at stages 1'–3', 1–3, and 5–6 are used to encrypt handshake traffic, i.e., for internal use.
- $\mathit{replay} = (\mathit{nonreplayable} : \{1', 2', 3', 4', 1, 2, 3, 4, 5, 6, 7, 8\})$: None of the keys is replayable. (Note that in contrast to the protocol with matching time periods, the server here rejects the initial four keys, making stages 1'–4' technically non-replayable.)

Session and Contributive Identifiers. Recall that $\mathtt{SC} \coloneqq \mathtt{ServerCertificate}$ denotes the server certificate $\mathit{cert}[pk_s]$. The session identifiers are

$$\mathit{sid}_{1'} = \begin{cases} (\text{“EHTS}'\text{”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}) \text{ if } \sigma.\mathit{role} = \mathit{initiator} \\ \bot \text{ otherwise,} \end{cases}$$

$$\mathit{sid}_{2'} = \begin{cases} (\text{“SHTS}'\text{”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{CKC}') \text{ if } \sigma.\mathit{role} = \mathit{initiator} \\ \bot \text{ otherwise,} \end{cases}$$

$$\mathit{sid}_{3'} = \begin{cases} (\text{“CHTS}'\text{”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{CKC}') \text{ if } \sigma.\mathit{role} = \mathit{initiator} \\ \bot \text{ otherwise,} \end{cases}$$

$$\mathit{sid}_{4'} = \begin{cases} (\text{“ETS}'\text{”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{CKC}') \text{ if } \sigma.\mathit{role} = \mathit{initiator} \\ \bot \text{ otherwise,} \end{cases}$$

$$\mathit{sid}_1 = (\text{“EHTS”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{SH}),$$
$$\mathit{sid}_2 = (\text{“SHTS”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{SH}, \mathtt{CKC}),$$
$$\mathit{sid}_3 = (\text{“CHTS”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{SH}, \mathtt{CKC}),$$
$$\mathit{sid}_4 = (\text{“ETS”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{SH}, \mathtt{CKC}),$$
$$\mathit{sid}_5 = (\text{“SAHTS”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{SH}, \mathtt{CKC}, \mathtt{CC}, \mathtt{SKC}),$$
$$\mathit{sid}_6 = (\text{“CAHTS”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{SH}, \mathtt{CKC}, \mathtt{CC}, \mathtt{SKC}),$$
$$\mathit{sid}_7 = (\text{“SATS”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{SH}, \mathtt{CKC}, \mathtt{CC}, \mathtt{SKC}, \mathtt{SPK}^*, \mathtt{EE}),$$
$$\mathit{sid}_8 = (\text{“CATS”}, \mathtt{SC}, \mathtt{CH}, \mathtt{SSKC}, \mathtt{SH}, \mathtt{CKC}, \mathtt{CC}, \mathtt{SKC}, \mathtt{SPK}^*, \mathtt{EE}, \mathtt{SF}).$$

[8] Note that Stages 2'–4', rejected by the server, are also implicitly server-authenticated. The focus is however on the stages that are accepted by both parties.

The parties set the contributive identifiers as follows.

- Upon sending the `SemiStaticKEMCiphertext` and `ClientKEMCiphertext′` message, the client sets $cid_i \leftarrow sid_i$ for $i = 1'$ resp. $i \in \{2', 3', 4'\}$. Upon sending (resp. receiving) `SemiStaticKEMCiphertext`, the client (resp. server) sets $cid_1 \leftarrow$ ("EHTS", `SC`, `CH`, `SSKC`).
- Upon receiving the `SemiStaticKEMCiphertext` message, since the time periods do not match, the server sets $cid_i \leftarrow sid_i \leftarrow \perp$ for $i \in \{1', 2', 3', 4'\}$.
- Upon sending (resp. receiving) the `ServerHello` message, the server (resp. client) sets $cid_1 \leftarrow sid_1$.
- All other contributive identifiers are set as $cid_i \leftarrow$ ("label", `SC`, `CH`, `SSKC`, `SH`) when the corresponding session identifier sid_i is set (where "label" is the label string from sid_i.

Remark. The fact that the contributive identifiers at all stages only include messages up to `SH` (in both the cases of matching and unmatching time periods) means that it is enough for messages up to `SH` to be honestly delivered to prove the secrecy of forward-secret stage keys (cf. Case (2) (a) in Definition 4.2), and also of delayed forward-secret keys if the adversary does not corrupt the server semi-static key (cf. Case (3) (b) (i) in Definition 4.2).

5.2 Security Proofs

This section proves the security of the Section 3 protocol in the model presented in Section 4.

Match Security. The following theorem formalizes the match security of the protocol.

Theorem 5.1 (Match Security). *Assuming* KEM_s, KEM_e *and* KEM_c *to respectively be* δ_s, δ_e *and* δ_c*-correct, the advantage of any adversary that makes at most* $n_\sigma \coloneqq q_{\mathsf{NewSession}}$ *queries to oracle* NewSession *in the match security game for the Section 3 protocol (in both the cases of matching and unmatching periods) is at most* $(2\delta_s + \delta_e + \delta_c)\, n_\sigma + 2^{-257} n_\sigma^2$.

Proof. It suffices to bound the probability that each of the winning conditions is satisfied. Note that the theorem statement does not impose any restriction on the computational power of the adversary; the match security of the protocol is thereby information theoretic.

1. *More than two sessions share the same identifier at some non-replayable stage.* At each non-replayable stage accepted by both parties, the session identifier includes the session's own random nonce (within the `ClientHello` or `ServerHello` message). So three pairwise distinct sessions can share the same identifier only if at least two of them sample the same nonce. By the birthday bound, the probability of this event is at most $\binom{n_\sigma}{2} 2^{-256} \leq 2^{-257} n_\sigma^2$.

2. *Two sessions share the same identifier at some stage but have non-opposite roles (except for potential multiple responders in replayable stages).* For non-replayable stages, assuming that at most two accepting sessions can share the same identifier (which is the case except with the above probability), no two responders or initiators can hold the same identifier since they never accept `ClientHello` and `ServerHello` messages typed with a non-opposite role. For the replayable stages (stages 1–4 in the matching period case), two initiators similarly share the same identifier only upon nonce collisions, bounded by the probability above. (There might be multiple responder sessions partnered in these stages 1–4, though, as the client's first flight of messages can indeed be replayed to several server sessions.)
3. *Two sessions share the same identifier at some stage but computed different stage keys.* The key a session computes at any stage is entirely determined by the messages it received up to the stage, and these are included in the session identifier. It follows that two partnered sessions can compute different keys only if the correctness of one of the KEMs fails. In a protocol execution, the participants together decapsulate one KEM_e ciphertext, one KEM_c ciphertext and either two KEM_s ciphertexts in the case of matching time periods or one in the case of unmatching time periods. The probability that two partnered sessions compute different stage keys is thus at most $(2\delta_s + \delta_e + \delta_c)\, n_\sigma$.
4. *Two sessions share the same identifier but have distinct or unspecified contributive identifiers at some stage.* By construction of the protocol, the final contributive identifier at any stage is always a substring to the session identifier once the session accepts the stage, so this cannot happen.
5. *Two distinct stages share the same session identifier.* This event cannot occur as each session identifier carries a unique label.
6. *A stage has (retroactively) reached explicit authentication, but the partnered session does not belong to the intended peer.* A server sessions learns the identity of the partner client through the `ClientCertificate` message which is included in the session identifier of the stage from which the server is explicitly authenticated. It thus guarantees that honest sessions with matching session identifiers agree on the client identity.
   A client session learns the server identity via the preloaded server certificate which is in all session identifiers. Partnered session identifiers thereby agree on the server identity. □

Multi-Stage Security. The following two theorems capture the multi-stage security of the protocol with matching and unmatching time periods.

Theorem 5.2 (Multi-Stage Security – Matching Time Periods). *Suppose that for*

$$(\mathsf{S}, \mathsf{A}) \in \left\{ \begin{array}{l} (\mathsf{KEM}_c, \text{IND-CCA}), (\mathsf{KEM}_s, \text{IND-CCA}), (\mathsf{KEM}_e, \text{IND-1CCA}), \\ (\mathsf{HKDF.Extract}, \text{PRF}), (\mathsf{HKDF.Extract}, \text{dual-PRF}), \\ (\mathsf{HKDF.Expand}, \text{PRF}), (\mathsf{HMAC}, \text{EUF-CMA}) \end{array} \right\}$$

S *is* $\left(T^{\mathsf{A}}_{\mathsf{S}}, q^{\mathsf{A}}_{\mathsf{S}}, \varepsilon^{\mathsf{A}}_{\mathsf{S}}\right)$-$\mathsf{A}$*-secure. Let* $\mathscr{A}$ *be an algorithm that runs in time at most* $T_{\mathscr{A}}$ *and makes at most* $q_{\mathscr{O}}$ *oracle queries for* $\mathscr{O} \in \{\mathsf{NewSession}, \mathsf{Send}, \mathsf{Reveal},$

$\mathsf{NextPeriod}, \mathsf{EndCurrentPeriod}, \mathsf{Corrupt}, \mathsf{SemiStaticCorrupt}, \mathsf{Test}\}$. *There exists a real constant* $\kappa \leq 1$ *such that if* $T_{\mathscr{A}} + q_{\mathsf{Send}} + q_{\mathsf{Test}} \leq \kappa \min_{(\mathsf{S},\mathsf{A})} \left(T_{\mathsf{S}}^{\mathsf{A}}\right)$ *and if* $n_\sigma \coloneqq q_{\mathsf{NewSession}} \leq \min_{(\mathsf{S},\mathsf{A})} \left(q_{\mathsf{S}}^{\mathsf{A}}\right)$*, then there exist (explicit) reduction algorithms to the respective* $\left(T_{\mathsf{S}}^{\mathsf{A}}, q_{\mathsf{S}}^{\mathsf{A}}, \varepsilon_{\mathsf{S}}^{\mathsf{A}}\right)$-$\mathsf{A}$ *security of* S *such that the advantage of* $\mathscr{A}$ *in the multi-stage security game in the case of matching time periods is at most*

$$
\begin{aligned}
&2^{-257} n_\sigma^2 + \varepsilon_H^{\text{Coll}} + (2\delta_s + \delta_e + \delta_c) n_\sigma \\
&+ 8 n_\sigma \left(\begin{array}{l}
n_{id} \left(\begin{array}{l} \varepsilon_{\mathsf{KEM}_c}^{\text{IND-CCA}} + \varepsilon_{\mathsf{KEM}_s}^{\text{IND-CCA}} \\ + 3\varepsilon_{\mathsf{HKDF.Extract}}^{\text{PRF}} + \varepsilon_{\mathsf{HKDF.Extract}}^{\text{dual-PRF}} \\ + 4\varepsilon_{\mathsf{HKDF.Expand}}^{\text{PRF}} + \varepsilon_{\mathsf{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \\
+ n_{id} \cdot n_{period} \cdot n_\sigma \left(\begin{array}{l} \varepsilon_{\mathsf{KEM}_s}^{\text{IND-CCA}} + 3\varepsilon_{\mathsf{HKDF.Extract}}^{\text{PRF}} \\ + \varepsilon_{\mathsf{HKDF.Extract}}^{\text{dual-PRF}} + 4\varepsilon_{\mathsf{HKDF.Expand}}^{\text{PRF}} \\ + \varepsilon_{\mathsf{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \\
+ n_\sigma \cdot \varepsilon_{\mathsf{KEM}_e}^{\text{IND-1CCA}}
\end{array} \right),
\end{aligned}
$$

with $n_{id} \coloneqq |ID|$, $n_{period} \coloneqq q_{\mathsf{NextPeriod}} + 1$, $\varepsilon_H^{\text{Coll}}$ *the probability that an algorithm given in the proof finds a collision for* H *by running* $\mathscr{A}$ *as subroutine, and* KEM_s, KEM_e, *and* KEM_c *being* δ_s-, δ_e-, *and* δ_c-*correct, respectively.*

Proof. The proof consists in a sequence of games that starts with the multi-stage security game and which are indistinguishable under the theorem assumptions. In the last game, the stage keys are uniformly random values and no session can maliciously accept a stage, i.e., the advantage of the adversary is nil.

Game 0. This is the multi-stage security game as in Definition 4.4.

Game 1. The challenger of this games returns 1 (i.e., the adversary wins the game) if there exist two distinct honest sessions with the same role that pick the same nonce. The advantage in distinguishing this game from the previous one is at most $\binom{n_\sigma}{2} 2^{-256} \leq 2^{-257} n_\sigma^2$.

Game 2. In this game, the challenger returns 1 if any two honest sessions compute the same hash value on different inputs to the hash function H. The advantage in distinguishing this game from the previous one is at most the probability that the challenger, with the adversary as subroutine, computes a collision for H. This collision can then be used by an algorithm that reduces the problem of distinguishing the two games to finding a collision for H.

Game 3. In this game, the challenger returns 1 if encapsulation and decapsulation between any two honest sessions yields a correctness error (as per Section 2.6). Assuming KEM_s, KEM_e, and KEM_c to be respectively δ_s-, δ_e-, and δ_c-correct, a KEM correctness error during the interaction with the adversary occurs with probability at most $(2\delta_s + \delta_e + \delta_c) n_\sigma$.

Focusing on a Single Target Session and Stage (σ, i). The next step restricts the adversary to target a single session σ in stage i, be it via a (single) Test query or via making that session maliciously accepts. These are later

referred to as the *target* session and stage (σ, i). This induces a loss of $8n_\sigma$ (accounting for the maximum number of stages across all sessions), though via two different arguments: restricting the adversary to a single Test query requires a hybrid argument following that of Dowling et al. [19], which is summarized in the following paragraphs. For malicious acceptance, this is a mere guessing step on (at least) one session and stage maliciously accepting, which succeeds with probability $1/8n_\sigma$. To enhance readability, both variants are treated as a combined game change, as the overall bound will coincide and the subsequent proof steps will argue along the same lines for either case.

The Hybrid Argument. For the hybrid argument, restricting the adversary to a single Test query, we follow [19, Appendix A]. More precisely, for an algorithm $\mathscr{A}$ as in the theorem statement, let $\mathscr{B}$ be an algorithm that interacts with the challenger of Game 2 and runs $\mathscr{A}$ as a subroutine. $\mathscr{B}$ chooses an integer $t \in [\![8n_\sigma]\!]$ uniformly at random at the beginning of the game; the range accounts for the maximum total number of stages across all sessions. For the first $t-1$ Test queries $\mathscr{A}$ makes, algorithm $\mathscr{B}$ returns the key computed by the tested session at the stage of the query. $\mathscr{B}$ forwards the t-th query from $\mathscr{A}$ to the challenger and for the remaining queries, $\mathscr{B}$ returns uniformly random keys. Dowling et al. showed [19, Appendix A] that (in their model,) the advantage of $\mathscr{B}$ is at least a fraction $8n_\sigma$ of the advantage of $\mathscr{A}$ via a standard hybrid argument.

However, the proof is not entirely trivial because oracles Send and Test overwrite the internal keys computed by sessions partnered at a tested stage, and session identifiers are defined by handshake messages in clear text. It means that $\mathscr{B}$ must be able to decrypt handshake messages to determine which sessions are partnered and properly emulate the game to $\mathscr{A}$, i.e., take into account the Test queries that $\mathscr{B}$ does not forward. To decrypt handshake traffic and identify potential partnered sessions, $\mathscr{B}$ submits additional Reveal queries for the internal keys of the session involved in the Send or Test query. Denoting by M_i the number of internal keys, that means at most $M_i\,(q_{\mathsf{Send}} + q_{\mathsf{Test}})$ additional Reveal queries are made. The crux of the matter is then to show that these additional Reveal queries do not cause the only Test query forwarded by $\mathscr{B}$ to be rejected when it would have otherwise been replied to. Although the model from Section 4 differs from the model of Dowling et al., the Send, Reveal and Test oracles are very similarly defined in the two models and the other oracles do not impact Test queries. The arguments of Dowling et al. thereby also apply in the present context.

In summary, $\mathscr{B}$ makes at most one Test query, at most $q_{\mathsf{Reveal}} + M_i\,(q_{\mathsf{Send}} + q_{\mathsf{Test}})$ Reveal queries and the same amount of queries as $\mathscr{A}$ to the remaining oracles, and the advantage of $\mathscr{B}$ is at least a fraction q_{Test} of the advantage of $\mathscr{A}$. Besides, considering AEAD encryption and decryption operations to be constant time, the runtime of $\mathscr{B}$ is of order $T_{\mathscr{A}} + O\,(q_{\mathsf{Send}} + q_{\mathsf{Test}})$. On this account, the advantage of the adversary can first be analyzed in a game restricted to a single Test query and then later extrapolated to a game with multiple queries.

Game 4. The adversary is restricted to a single *target* session and stage (σ, i), guessed at the outset of the game, and made to lose if it would otherwise win the game via (1) testing and distinguishing another session/stage key or (2) malicious acceptance of another session/stage.
By the hybrid argument above, for part (1), there exists an algorithm that runs the adversary as subroutine and has an advantage of at least a $8n_\sigma$ fraction of the advantage of $\mathscr{A}$ in the previous game by making at most one Test query, $q_{\mathsf{Reveal}} + M_i\,(q_{\mathsf{Send}} + q_{\mathsf{Test}})$ Reveal queries and the same amount of queries as $\mathscr{A}$ for the other oracles. Besides, the algorithm runs in time $T_{\mathscr{A}} + O\,(q_{\mathsf{Send}} + q_{\mathsf{Test}})$.
Similarly, for the case of malicious acceptance, (σ, i) coincides with the (at least) one maliciously accepting session with probability at least $1/8n_\sigma$. Overall, this hence introduces a factor of $8n_\sigma$.

Case Distinction Based on Freshness. For the now unique session (σ, i) targeted by the adversary (via Test or malicious acceptance), which now is known at the outset of the game, two main cases are distinguished next:

A. $i \leq 4$, i.e., the adversary targets a stage key that achieves *delayed forward secrecy*, and
B. $i \geq 5$, i.e., the adversary targets a stage key that achieves *forward secrecy*.

These cases are further subdivided into the following sub-cases, closely corresponding to the conditions for the targeted session to be fresh, cf. Definition 4.2. (Note that while freshness is a prerequisite only for testing a session, ruling out malicious acceptance of the targeted session will follow almost identical arguments.)

A.I. Server implicit authentication:
  (a) $\sigma.role = initiator$, $i \geq 2$, and $sk_{\sigma.pid}$ is not corrupt (recall that the server is implicitly authenticated from stage 2 on).
  (b) $\sigma.role = initiator$ and $sk_{\sigma.pid}^{\sigma.period}$ is not corrupt.

A.II. (a) Honest contributive initiator: $\sigma.role = responder$ and there exists a session $\sigma' \neq \sigma$ such that $\sigma.cid_i = \sigma'.cid_i$ and $\sigma.role \neq \sigma'.role$, and $sk_{\sigma.id}^{\sigma.period}$ is not corrupt.
  (b) Delayed forward secrecy: $\sigma.role = responder$, $\sigma.status_8 = accepted$, the adversary did not corrupt $sk_{\sigma.pid}$ before σ accepted stage 8 and $sk_{\sigma.id}^{\sigma.period}$ is not corrupt.

B.I. Honest contributive partner: There exists a session $\sigma' \neq \sigma$ such that $\sigma.cid_i = \sigma'.cid_i$ and $\sigma.role \neq \sigma'.role$.

B.II. Implicit authentication:
  (a) $\sigma.role = initiator$, $sk_{\sigma.pid}$ is not corrupt (recall that $i \geq 2 = \sigma.iauth_s$ in case B), and $\sigma.status_7 \neq accepted$.
  (b) $\sigma.role = responder$ and $sk_{\sigma.pid}$ is not corrupt (recall that $i \geq 5 = \sigma.iauth_c$ in case B) and $\sigma.status_8 \neq accepted$.

B.III. Forward secrecy: There is no session $\sigma' \neq \sigma$ such that $\sigma.cid_i = \sigma'.cid_i$ and $\sigma.role \neq \sigma'.role$, and
   (a) $\sigma.role = initiator$, $\sigma.status_7 = accepted$ and the adversary did not corrupt $sk_{\sigma.pid}$ before σ accepted stage 7.
   (b) $\sigma.role = responder$, $\sigma.status_8 = accepted$ and the adversary did not corrupt $sk_{\sigma.pid}$ before σ accepted stage 8.

Note that Case A.II omits client implicit authentication from Definition 4.2 since the client is not yet implicitly authenticated at stages 1–4. Besides, the sub-cases of Case B.II exclude that the stage reached explicit partner authentication, although the corresponding sub-cases in Definition 4.2 do not. This is simply because Case B.III already handles the situation in which a stage reached explicit partner authentication but the adversary never corrupted the partner's long-term key throughout the interaction (and a fortiori not before the stage reached explicit partner authentication). Avoiding this overlap between Cases B.II and B.III eases the argumentation, and a consequence is that Case B.II need not consider malicious acceptance in the analysis.

Case A.I.a: Targeted Client with Non-compromised Partner Long-Term Key

This case is limited to the target stages 2–4. The secrecy of the client stage key relies on the indistinguishability of the KEM_s ciphertext C_s since the server is implicitly authenticated from stage 2 on.

Game A.I.a.1 (Guess Partner Identity). The challenger guesses at the beginning of the game the identity of the intended partner of the target session and aborts and returns 0 if the guess is incorrect. This decreases the advantage of the adversary by a factor at most n_{id}.

Game A.I.a.2 (Server Long-Term KEM). In this game, K_s is replaced in σ with a uniformly random value. In any session of $\sigma.pid$ that receives the client ciphertext, K_s is replaced with the same value (no KEM correctness error occurs by Game 3). (Note that this includes potential responder sessions of $\sigma.pid$ receiving replayed copies of the initial messages of σ.)
Distinguishing this game from the previous one can be reduced to the IND-CCA security of KEM_s as follows. The reduction algorithm, upon receiving the challenge tuple $(pk^\star, C^\star, K^\star)$, first sets $pk^\star$ as the server public key of $\sigma.pid$. Then, it sets $C^\star$ as the ciphertext C_s in the `ClientKEMCiphertext` message of the target session and uses $K^\star$ as K_s. For any session of $\sigma.pid$, if it receives C_s then $K^\star$ is used as K_s. If it receives any other ciphertext, then the reduction algorithm makes a decapsulation query and uses the returned value as K_S for that session. The reduction algorithm eventually forwards the decision bit of the adversary to the IND-CCA challenger.

Game A.I.a.3 (HS**).** The handshake secret HS in σ is now replaced with a uniformly random value. In any session of $\sigma.pid$ that received the ciphertext C_s sent by σ (possibly via a full replay of the client's initial messages), the

handshake secret HS is consistently replaced with uniformly random values. We maintain consistency across all the sessions that share the same derived early secret dES.

Distinguishing this game from the previous is reducible to the dual PRF security of HKDF.Extract. It suffices for the reduction algorithm to make oracle queries on PRF labels dES (K_s is tacitly set as the key of the dual PRF challenger) to compute HS for σ and any session of $\sigma.pid$ that received the C_s sent by σ, and forward the decision bit of the adversary.

Note that if a session is not partnered at stage i but did receive the ciphertext C_s send by σ, then even though the key is computed at stage i can be tested, the latter is independent from the one computed by σ.

Game A.I.a.4 (SHTS**,** CHTS**,** ETS **and** dHS**).** In σ and any session of $\sigma.pid$ that received C_s, the derivation of the handshake-traffic secrets SHTS and CHTS, early-traffic secret ETS, and derived handshake secret dHS from the now-random HS in σ is replaced by a random function. This in particular replaces SHTS, CHTS, ETS and dHS in σ with independent, uniformly random values. If the σ's initial messages were not altered, then these secrets are the same in matching responder sessions (possibly several, through replays) as the ones used in σ, otherwise they are independent random values (recall that the game halts if the hashes of two distinct values collide).

Distinguishing this game change can be reduced to the PRF security of HKDF.Expand, keyed with the previously replaced HS value.

Game A.I.a.5 (IMS**).** In σ and any session of $\sigma.pid$ that received C_s, the derivation of the intermediate master secret IMS from the now-random dHS in σ is replaced by a random function. This in particular replaces IMS in σ with a uniformly random value. The PRF security of HKDF.Extract allows to argue for the indistinguishability between this game and the previous one.

Game A.I.a.6 (dIMS**).** In σ and any session of $\sigma.pid$ that received C_s, the derivation of the derived intermediate master secret dIMS from the now-random IMS in σ is replaced by a random function. This in particular replaces dIMS in σ with a uniformly random value. The indistinguishabilty from the previous game relies again on the PRF security of HKDF.Expand.

Game A.I.a.7 (MS**).** In σ and any session of $\sigma.pid$ that received C_s, the derivation of the master secret MS from the now-random dIMS in σ is replaced by a random function. This in particular replaces MS in σ with a uniformly random value. The PRF security of HKDF.Extract is once again invoked to argue that this game is indistinguishable from the previous one.

Game A.I.a.8 (SAHTS**,** CAHTS**,** fk_s**,** SATS**,** fk_c **and** CATS**).** In σ and any session of $\sigma.pid$ that received C_s, the derivation of the values SAHTS, CAHTS, fk_s, SATS, fk_c and CATS from the now-random master secret MS in σ is replaced by a random function. This in particular replaces all these values in σ with independent, uniformly random values. The PRF security of HKDF.Expand implies the indistinguishability between this game and the previous.

Game A.I.a.9 (MAC Forgery). The challenger of this game, running the targeted client session σ, rejects the `ServerFinished` message in case σ

has no partner session at stage i. The fact that there is no partner session at stage i implies that no honest session computed a MAC tag on the transcript of the targeted client session (of which sid_i is a part). In other words, the adversary forged a MAC value and distinguishing this game from the previous one is therefore reducible to the EUF-CMA security of HMAC.

Note that since the challenger rejects the `ServerFinished` message in case σ has no partner at stage i, the event of malicious acceptance in which the client session accepts stage i without a partner session never occurs. Besides, all stage keys in the target session σ are uniformly random and independent from any non-partnered session, making them non-exposable via Reveal oracle queries. The advantage of the adversary in the last game is therefore nil.

It follows that the advantage of the adversary in Game 4 is in this case at most

$$n_{id}\left(\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s} + \varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}} + 2\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}} + 3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\right).$$

Case A.I.b: Targeted Client with Non-compromised Partner Semi-static Key

This case is fairly similar to the previous one, except that the secrecy of the stage keys now flows from the semi-static key contribution $K_s^{t_{s,c}}$.

Game A.I.b.1 (Guess Partner Identity and Time Period). At the beginning of the game, the challenger guesses the partner identity $\sigma.pid$ of the target session σ as well as the time period $\sigma.period$ (of the semi-static key) used in that session and aborts and returns 0 if any of these guesses is incorrect. This decreases the advantage of the adversary by a factor at most $n_{id} \cdot n_{period}$.

Game A.I.b.2 (Semi-static KEM). Similar to Game A.I.a.2, the encapsulated KEM key in σ is replaced with a uniformly random value, this time targeting $C_s^{t_{s,c}}$ and $K_s^{t_{s,c}}$ instead of C_s and K_S. The reduction algorithm here sets the challenge KEM public key $pk^\star$ as the semi-static key of $\sigma.pid$ in $\sigma.period$ (which it knows at the outset of the game based on the guesses above).

Note: A client session saves $pk'_{\sigma.pid,\sigma.id}$ received in a `SPK` message (during the transition from $\sigma.period$ to $\sigma.period + 1$) only after accepting stage 7. The server at this point is explicitly authenticated, and so the public semi-static key is guaranteed to come from an honest partner session. This is important for the protocol's security. Indeed, if the client were ever to use a semi-static key that was saved before verifying the `ServerFinished` message, an attacker could inject its own semi-static key, and later compute all stage keys by corrupting the long-term secret key of the server.

Game A.I.b.3 (ES**).** The early secret ES is now replaced in σ with a uniformly random value. The same value is used in any session of $\sigma.pid$ that received

the ciphertext $C_s^{t_{s,c}}$ sent by σ (possibly via a full replay of the client's initial messages).

Distinguishing this game from the previous is reducible to the dual PRF security of HKDF.Extract. It suffices for the reduction algorithm to make one oracle query on 0 (with K_s^t as the key of the dual PRF challenger), set the value as ES in σ and any session of $\sigma.pid$ that received $C_s^{t_{s,c}}$, and forward the decision bit of the adversary.

Game A.I.b.4 (EHTS, dES). The challenger replaces the early handshake traffic secret EHTS and the derived early secret dES in σ with uniformly random values. The same values are used in any session of $\sigma.pid$ that received the ciphertext $C_s^{t_{s,c}}$ sent by σ and is hence using the same value ES. Distinguishing this game from the previous can be reduced to the PRF security of HKDF.Expand.

Game A.I.b.5 (HS). In σ and any session of $\sigma.pid$ that received $C_s^{t_{s,c}}$, the derivation of the handshake secret HS from the now-random dES in σ is replaced by a random function. This in particular replaces HS in σ with a uniformly random value. Now the PRF security of HKDF.Extract serves as argument for the indistinguishability from the previous game.

Games A.I.b.6–A.I.b.11. Analogous to Games A.I.a.4–A.I.a.9, maintaining consistency in sessions of $\sigma.pid$ that received $C_s^{t_{s,c}}$.

In Case A.I.b, the advantage of the adversary in Game 4 is therefore at most

$$n_{id} \cdot n_{period} \left(\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s} + \varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}} + 3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}} + 4\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\right).$$

Case A.II.a: Targeted Server with Honest Contributive Partner and Non-compromised Semi-static Key

The existence of an honest contributive partner and the fact that the server semi-static key is not compromised ensure that the adversary cannot recover the semi-static value used in the computation of the key.

Note that since, by Game 1, no two sessions with the same role can share the same nonce, there is exactly one initiator session that shares the same contributive identifier at stage i.[9] Denote this session by σ'.

Game A.II.a.1 (Guess Server Identity, Period, Contributive Partner). The challenger guesses at the beginning of the game the owner identity $\sigma.id$ of the target session σ, the time period $\sigma.period$ (of the semi-static key) used in that session, as well as the contributive partner session σ' of σ and aborts and returns 0 if any of these guesses is incorrect. This decreases the advantage of the adversary by a factor at most $n_{id} \cdot n_{period} \cdot n_\sigma$.

Game A.II.a.2 (Semi-static KEM). In this game, the challenger replaces the semi-static value $K_s^{t_{s,c}}$ encapsulated in ciphertext $C_s^{t_{s,c}}$ in both σ and σ'

[9] While replays at stage $i \leq 4$ are possible on the responder side, initiator sessions are unique (cf. Match security, Definiton 4.1, condition 2).

with a uniformly random value (the same); the same value is used in other sessions of $\sigma.id$ in time period $\sigma.period$ receiving the same ciphertext $C_s^{t_{s,c}}$ – no KEM correctness error occurs by assumption.

Distinguishing this game to the previous one can be reduced to the IND-CCA security of KEM_s as follows. The reduction algorithm, upon receiving the challenge tuple $(pk^\star, C^\star, K^\star)$, first sets $pk^\star$ as the server semi-static public key of $\sigma.id$ in $\sigma.period$. Then, it sets $C^\star$ as the ciphertext $C_s^{t_{s,c}}$ in the `ClientHello` messages of σ'. Since $\sigma.cid_i = \sigma'.cid_i$, the ciphertext received by σ is the same as the one σ' sent. The reduction algorithm can thus replace $K_s^{t_{s,c}}$ with $K^\star$ in both σ and σ', as well as any other sessions of $\sigma.id$ in time period $\sigma.period$ receiving the same ciphertext $C^\star$. It uses its decapsulation oracle to decapsulate any other ciphertexts for the public key $pk^\star$ of $\sigma.id$ in period $\sigma.period$. The reduction algorithm eventually forwards the decision bit of the adversary to the IND-CCA challenger.

Games A.II.a.3–A.II.a.10. Similar to Games A.I.b.3–A.I.b.10.

Game A.II.a.11 (MAC Forgery). The challenger of this game, running the targeted server session, rejects the `ClientFinished` message in case there is no partner session at stage 8. The fact that there is no partner session at stage 8 implies that no honest session computed a MAC tag on the transcript of the targeted server session. In other words, the adversary forged a MAC value and distinguishing this game from the previous one is therefore reducible to the EUF-CMA security of HMAC.

The advantage of the adversary in Game 4 is in this case at most

$$n_{id} \cdot n_{period} \cdot n_\sigma \left(\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s} + \varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}} + 3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}} + 4\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\right).$$

Case A.II.b: Targeted Server with Explicitly Authenticated Client and Non-compromised Semi-static Key

The fact that the client's secret key is not compromised before stage 7 is accepted allows to distinguish two cases: either the targeted session accepts at stage 7 with an honest partner, in which case the conditions for the prior Case A.II.a are satisfied; or the targeted session accepts without an honest partner, in which case *up to this point*, the MAC key of the client, used to compute the `CF` message that made the target session accept, is secret. For the latter case, the secrecy of the MAC key is established similar to Case A.I.a, but is based on the client long-term KEM key K_c, the subsequent sequence ($K_c \rightarrow \text{MS} \rightarrow fk_c$) of key derivations and the unforgeability of the MAC.

Game A.II.b.1 (Guess Partner Identity). Identical to Game A.I.a.0.

Game A.II.b.2 (Client KEM). Similar to Game A.I.a.1, but with K_c and C_c instead of K_s and C_s. With K_c now replaced by a uniformly random value, the next step is the derivation of the master secret.

Game A.II.b.3 (MS**).** In σ and any session of $\sigma.pid$ that received C_c, the derivation of the master secret MS from the now-random K_c in σ is replaced by a random function. This in particular replaces MS in σ with a uniformly random value.

Distinguishing this game from the previous is reducible to the dual PRF security of HKDF.Extract. It suffices for the reduction algorithm to make oracle queries on the with labels dIMS (K_c is tacitly set as the key of the dual PRF challenger) to compute MS for σ and any session of $\sigma.pid$ that received the C_c sent by σ, and forward the decision bit of the adversary.

Note that if a session is not partnered at stage i but did receive the ciphertext C_c send by σ, then even though the key is computed at stage i can be tested, the latter is independent from the one compute by σ.

Game A.II.b.4 (SAHTS**,** CAHTS**,** fk_s**,** SATS**,** fk_c **and** CATS**).** In σ and any session of $\sigma.pid$ that received C_c, the derivation of the values SAHTS, CAHTS, fk_s, SATS, fk_c and CATS from the now-random master secret MS in σ is replaced by a random function. This in particular replaces all these values in σ with independent, uniformly random values. The same values as in σ are used for such sessions if they are partnered at stage i, and they are otherwise replaced with independent uniformly random values.

The PRF security of HKDF.Expand guarantees that this game is indistinguishable from the previous.

Game A.II.b.5 (MAC Forgery). The challenger of this game, running the targeted server session, rejects the `ClientFinished` message in case there is no partner session at stage 8. The fact that there is no partner session at stage 8 implies that no honest session computed a MAC tag on the transcript of the targeted server session. In other words, the adversary forged a MAC value and distinguishing this game from the previous one is therefore reducible to the EUF-CMA security of HMAC.

In the second sub-case, the advantage of the adversary in Game 4 is hence at most

$$n_{id}\left(\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_c} + \varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}} + \varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\right).$$

Case B.I: Honest Contributive Partner

For keys with expected forward secrecy, the existence of an honest contributive partner at the targeted stage i guarantees that the adversary does not have access to the ephemeral value generated by the server. The secrecy of these stage keys therefore mainly relies on the indistinguishability of the ephemeral KEM.

As no two honest sessions with the same role share the same nonce, there exists exactly one contributive partner at stage i when σ accepts.[10] Denote it by

[10] As $i \geq 5$, the targeted stage is non-replayable. Furthermore, a targeted server session always has a unique initiator partner, even if the clients' message are replayed, and a targeted client has a unique responder partner due to the inclusion of the `ServerHello` message in the contributive identifiers.

σ'. Let $\sigma_c \in \{\sigma, \sigma'\}$ be such that $\sigma_c.role = \mathit{initiator}$ and $\sigma_s \in \{\sigma, \sigma'\}$ such that $\sigma_s.role = \mathit{responder}$.

Game B.I.1 (Guess Contributive Session). The challenger guesses at the beginning of the game the contributive partner session σ' of the target session σ and aborts and returns 0 if the guess is incorrect. This decreases the advantage of the adversary by a factor at most n_σ.

Game B.I.2 (Ephemeral KEM) The challenger of this game replaces the ephemeral secret value K_e in σ_s with a uniformly random value, and uses the same value in σ_c if the latter receives the ciphertext C_e that σ_s sent (no KEM correctness error occurs by assumption).

Distinguishing this game from the previous one can be reduced to the IND-1CCA security of KEM_e as follows. Upon receiving the challenge tuple ($pk^\star$, $C^\star, K^\star$), the reduction algorithm sends $pk^\star$ as the ephemeral public key pk_e in the `ClientHello` message. Since σ_c and σ_s are contributively partnered, they agree on `ClientHello` and this ephemeral public key is thus delivered to σ_s. The reduction algorithm then uses $C^\star$ as the ephemeral ciphertext C_e in the `ServerHello` message and $K^\star$ as the ephemeral secret K_e in σ_s. If $C^\star$ is delivered to σ_c, then the reduction algorithm also uses $K^\star$ in σ_c, otherwise it makes a (single) decapsulation query to the IND-1CCA challenger on the received ciphertext and uses the returned value as ephemeral secret. The decision bit of the adversary is ultimately forwarded to the IND-1CCA challenger.

Game B.I.3 (IMS**).** In σ_s (and σ_c if it received the ciphertext C_e computed by σ_s), the derivation of the intermediate master secret IMS from the now-random K_e in σ_s is replaced by a random function. This in particular replaces IMS in σ_s with a uniformly random value. The indistinguishability of this game from the previous can be reduced to the dual PRF security of $\mathsf{HKDF.Extract}$ where the key of the dual PRF challenger is implicitly set as K_e.

Game B.I.4 (dIMS**).** In σ_s (and σ_c if it received the ciphertext C_e computed by σ_s), the derivation of the derived intermediate master secret dIMS from the now-random IMS in σ_s is replaced by a random function. This in particular replaces dIMS in σ_s with a uniformly random value. The PRF security of $\mathsf{HKDF.Expand}$ substantiates the indistinguishability of this game from the previous one.

Game B.I.5 (MS**).** In σ_s (and σ_c if it received σ_s's ciphertext C_e), the derivation of the master secret IMS from the now-random dIMS in σ_s is replaced by a random function. This in particular replaces MS in σ_s with a uniformly random value. The PRF security of $\mathsf{HKDF.Extract}$ justifies the indistinguishability from the previous game.

Game B.I.6 (SAHTS**,** CAHTS**,** fk_s**,** SATS**,** fk_c **and** CATS**).** In σ_s (and σ_c if it received the ciphertext C_e computed by σ_s), the derivation of the values SAHTS, CAHTS, fk_s, SATS, fk_c and CATS from the now-random master secret MS in σ_s is replaced by a random function. This in particular replaces all these values in σ_s with independent and uniformly uniformly random

values. Note that if σ_c and σ_s are not partnered, then the adversary may query the stage keys computed by σ_c, but these keys are independent from those computed by σ_s due to the (hashed) session identifiers entering the key derivation. The indistinguishability from the previous game follows from the PRF security of $\mathsf{HKDF.Expand}$.

Game B.I.7 (MAC Forgery). The challenger of this game rejects the received finished message in the target session in case there is no partner session at the stage of explicit authentication (7 if $\sigma.role = initiator$ and 8 if $\sigma.role = responder$). The fact that there is no partner session at the latter stage implies that no honest session computed a MAC tag on the transcript of the targeted session. In other words, the adversary forged a MAC value and distinguishing this game from the previous one is therefore reducible to the EUF-CMA security of HMAC.

The advantage of the adversary in Game 4 is thus in this case at most

$$n_\sigma \left(\varepsilon^{\text{IND-1CCA}}_{\mathsf{KEM}_e} + \varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}} + \varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}} + 2\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\right).$$

Case B.II.a: Targeted Client with Implicitly Authenticated Server

The secrecy of these keys rely on the indistinguishability of the KEM_s ciphertext C_s encapsulating the server long-term secret. This case is handled identically to the case A.I.a with the exception that malicious acceptance is not a concern in this case since the targeted session did not accept stage 4, i.e., Games B.II.a.1–B.II.a.8 are identical to Game A.I.a.1–A.I.a.8.

In this case, the advantage of the adversary in Game 4 is at most

$$n_{id} \left(\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s} + \varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}} + 2\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}} + 3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}}\right).$$

Case B.II.b: Targeted Server with Implicitly Authenticated Client

The security of the KEM_c ciphertext C_c supports the secrecy of the stage keys in this case. This case is similar to Case A.II.b, except that malicious acceptance can once again not occur. The advantage of the adversary in Game 4 is in this case at most

$$n_{id} \left(\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_c} + \varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}} + \varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}}\right).$$

Case B.III.a: Targeted Client with Explicitly Authenticated Server, No Contributive Partner

A crucial difference between this case and case B.II.a is that the adversary may now corrupt the server long-term key *after* the server is explicitly authenticated.

This a priori raises an issue in the replacement of the server long-term KEM since the reduction would not be able to return the challenge secret key corresponding to the public key that it set as the server public key.

The main idea is now to show that the adversary cannot make the client maliciously accept stage 7 without having first corrupted the server long-term key if the MAC used to compute the `ServerFinished` message is secure. In other words, if the client accepts stage 7 (which is an assumption of Case B.III.a), only one of two things can occur: either there exists an honest contributive partner (which is excluded in Case B.III.a and was dealt with in Case B.I) or the adversary computed a valid MAC forgery. As this case excludes the existence of an honest contributive partner, it suffices to bound by above the probability that the targeted client maliciously accepts stage 7.

Games B.III.a.1–B.III.a.11. Game B.III.a.1 is identical to Game B.II.a.1 and Games B.III.a.2–B.III.a.10 are defined similarly to Games B.II.a.2–B.II.a.10, except that the challenger aborts the interaction with the adversary and returns 1 (i.e., the adversary wins) if the targeted client session accepts stage 7 without a partner session. The reason behind this abort is that the adversary could corrupt the server long-term key in Game B.III.a.1, which was not possible in Game B.II.a.1, but the reduction algorithm would not be able to return the KEM long-term key to the adversary. However, the corruption of the server long-term key cannot occur before the target client session accepts stage 7 by definition of Case B.III.a, which means that the adversary would first have to send a valid MAC forgery.
Game B.III.a.11 is identical to Game A.I.a.11. In the last game, the client session never accepts stage 7 without a partner session.

The adversary can thus make the targeted client maliciously accept with probability at most

$$n_{id}\left(\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s} + \varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}} + 3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}} + 4\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\right).$$

Therefore, if the targeted client session accepts stage 7, it is guaranteed that except with the above probability, it has an honest partner at that stage (and only that partner could recover the ephemeral value), and thus also at all prior stages given how session identifiers are defined. A consequence is that stages from 1 to 7 are all explicitly authenticated (retroactively for stages 1 to 6), and the keys computed at these stages are indistinguishable from random (even if the server long-term key is compromised after explicit authentication) according to the analysis of case B.I.

Case B.III.b: Targeted Server with Explicitly Authenticated Client, No Contributive Partner

The same reasoning as for client sessions applies here, except that the client is explicitly authenticated from stage 8 and that the parallel is now with Case B.II.b. The advantage of the adversary in Game 4 is in this case at most

$$n_{id}\left(\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_c}+\varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}}+\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}}+\varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\right).$$

Overall Advantage

The advantage of the adversary in Game 4 is the maximum of its advantage in each of the cases, which is upper-bounded by

$$\begin{aligned}
&n_{id}\begin{pmatrix}\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_c}+\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s}\\ +3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}}+\varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}}\\ +4\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}}+\varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\end{pmatrix}\\
&+n_{id}\cdot n_{period}\cdot n_\sigma\begin{pmatrix}\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s}+3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}}\\ +\varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}}+4\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}}\\ +\varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\end{pmatrix}\\
&+n_\sigma\cdot\varepsilon^{\text{IND-1CCA}}_{\mathsf{KEM}_e}.
\end{aligned}$$

Therefore, the advantage of the adversary in the multi-stage security game (Game 0) is at most

$$\begin{aligned}
&2^{-257}n_\sigma^2+\varepsilon^{\text{Coll}}_H+(2\delta_s+\delta_e+\delta_c)n_\sigma\\
&+8n_\sigma\begin{pmatrix}n_{id}\begin{pmatrix}\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_c}+\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s}\\ +3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}}+\varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}}\\ +4\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}}+\varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\end{pmatrix}\\
+n_{id}\cdot n_{period}\cdot n_\sigma\begin{pmatrix}\varepsilon^{\text{IND-CCA}}_{\mathsf{KEM}_s}+2\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Extract}}\\ +\varepsilon^{\text{dual-PRF}}_{\mathsf{HKDF.Extract}}+3\varepsilon^{\text{PRF}}_{\mathsf{HKDF.Expand}}\\ +\varepsilon^{\text{EUF-CMA}}_{\mathsf{HMAC}}\end{pmatrix}\\
+n_\sigma\cdot\varepsilon^{\text{IND-1CCA}}_{\mathsf{KEM}_e}\end{pmatrix}.
\end{aligned}$$

, □

Theorem 5.3 (Multi-Stage Security – Unmatching Time Periods). *Suppose that for*

$$(\mathsf{S},\mathsf{A})\in\left\{\begin{matrix}(\mathsf{KEM}_c,\text{IND-CCA}),(\mathsf{KEM}_s,\text{IND-CCA}),(\mathsf{KEM}_e,\text{IND-1CCA}),\\ (\mathsf{HKDF.Extract},\text{PRF}),(\mathsf{HKDF.Extract},\text{dual-PRF}),\\ (\mathsf{HKDF.Expand},\text{PRF}),(\mathsf{HMAC},\text{EUF-CMA})\end{matrix}\right\}$$

S *is* $\left(T^{\mathsf{A}}_{\mathsf{S}},q^{\mathsf{A}}_{\mathsf{S}},\varepsilon^{\mathsf{A}}_{\mathsf{S}}\right)$-$\mathsf{A}$*-secure. Let* $\mathscr{A}$ *be an algorithm that runs in time at most* $T_{\mathscr{A}}$ *and makes at most* $q_{\mathscr{O}}$ *oracle queries for* $\mathscr{O}\in\{\mathsf{NewSession},\mathsf{Send},\mathsf{Reveal},\mathsf{NextPeriod},\mathsf{EndCurrentPeriod},\mathsf{Corrupt},\mathsf{SemiStaticCorrupt},\mathsf{Test}\}$. *There exists a real constant* $\kappa\le 1$ *such that if* $T_{\mathscr{A}}+q_{\mathsf{Send}}+q_{\mathsf{Test}}\le\kappa\min_{(\mathsf{S},\mathsf{A})}\left(T^{\mathsf{A}}_{\mathsf{S}}\right)$ *and if* $n_\sigma :=$

$q_{\mathsf{NewSession}} \leq \min_{(\mathsf{S},\mathsf{A})} \left(q_{\mathsf{S}}^{\mathsf{A}}\right)$, *then there exist (explicit) reduction algorithms to the respective* $\left(T_{\mathsf{S}}^{\mathsf{A}}, q_{\mathsf{S}}^{\mathsf{A}}, \varepsilon_{\mathsf{S}}^{\mathsf{A}}\right)$-$\mathsf{A}$ *security of* S *such that the advantage of* $\mathscr{A}$ *in the multi-stage security game in the case of matching time periods is at most*

$$
\begin{aligned}
&2^{-257} n_\sigma^2 + \varepsilon_H^{\text{Coll}} + (2\delta_s + \delta_e + \delta_c) n_\sigma \\
&+ 12 n_\sigma \left(\begin{array}{l}
n_{id} \left(\begin{array}{l} \varepsilon_{\mathsf{KEM}_c}^{\text{IND-CCA}} + \varepsilon_{\mathsf{KEM}_s}^{\text{IND-CCA}} \\ + 3\varepsilon_{\mathsf{HKDF.Extract}}^{\text{PRF}} + \varepsilon_{\mathsf{HKDF.Extract}}^{\text{dual-PRF}} \\ + 4\varepsilon_{\mathsf{HKDF.Expand}}^{\text{PRF}} + \varepsilon_{\mathsf{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \\
+ n_{id} \cdot n_{period} \cdot n_\sigma \left(\begin{array}{l} \varepsilon_{\mathsf{KEM}_s}^{\text{IND-CCA}} + \varepsilon_{\mathsf{HKDF.Extract}}^{\text{PRF}} \\ + \varepsilon_{\mathsf{HKDF.Extract}}^{\text{dual-PRF}} + 3\varepsilon_{\mathsf{HKDF.Expand}}^{\text{PRF}} \\ + \varepsilon_{\mathsf{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \\
+ n_\sigma \cdot \varepsilon_{\mathsf{KEM}_e}^{\text{IND-1CCA}}
\end{array} \right),
\end{aligned}
$$

with $n_{id} \coloneqq |ID|$, $n_{period} \coloneqq q_{\mathsf{NextPeriod}} + 1$, $\varepsilon_H^{\text{Coll}}$ *the probability that an algorithm given in the proof finds a collision for* H *by running* $\mathscr{A}$ *as subroutine, and* KEM_s, KEM_e, *and* KEM_c *being* δ_s-, δ_e-, *and* δ_c-*correct, respectively.*

Proof. The proof is similar to the proof of Theorem 5.2, except that there are twelve stages, i.e., four more stage keys due to rejecting EHTS′, SHTS′, CHTS′, ETS′, hence the factor $12n_\sigma$ when focusing on a single target session and stage (σ, i).

Compared to the case of matching time periods, there is one less call to HKDF.Extract and to HKDF.Expand in the key schedule for the keys that satisfy full forward secrecy. This however does not affect the overall upper bound as stages 1'–4', rejected by the server but accepted by the client, must still be accounted for. Note also that SHTS and CHTS are expanded from HS like dHS, so no additional hop must be introduced for those and the rest of the bound remains unaffected. □

6 Discussion

Identity Protection. TLS 1.3 protects parties' identities by following the SIGMA-I key exchange pattern of Krawczyk [32]. More specifically, it protects the server identity against passive attackers and the client identity against active attackers, the latter identity being revealed only after having seen a valid server signature. The KEMTLS protocol [46] carefully mimics these properties, achieving identity protection for the server against passive attackers and for the client against active attackers. Client identity protection in KEMTLS comes with an additional half or full round trip (depending on the targeted authentication properties). The KEMTLS-PDK protocol [47], in reducing roundtrips, sends the KEM encapsulation against the server's static key in cleartext. Unless an anonymous KEM [4, 27, 40] is deployed, this value might leak information about the server's identity.

Our protocol leverages the pre-loaded server certificate to reduce handshake round trips *while* achieving stronger identity protection: it protects both server and client identities against *active* attackers, both with delayed forward secrecy through encrypting the client certificate and `ClientKEMCiphertext` C_s under the server's semi-static key (authenticated in a previous handshake).

On the Security Proofs. The security proofs are similar to those of the KEMTLS protocol, are given in the standard model and do not rely on any form of adversary rewinding. Existing techniques in the literature (e.g., Song's "lifting lemma" [49]) can thus be used to prove the protocol secure against quantum adversaries as long as the underlying primitives are.

However, the proofs are non-tight (with the precise losses spelled out in exact-security terms) as they require to guess the test session as well as, depending on the proof case, the contributive session or the identity of the intended peer. The proofs can thus be understood as heuristic arguments for the soundness of the protocol design. It is worth noting that except for very recent work on TLS 1.3 [16, 17], most proofs of deployed authenticated key-exchange protocols are also non-tight.

Downgrade Resilience. The model in Section 4 does not capture algorithm negotiation although any practical deployment of the protocol would support multiple instantiations for each primitive. However, one can still informally argue that the downgrade resilience properties of the protocol in Section 3 are similar to those of the KEMTLS protocol. More precisely, an active adversary could in principle make a party choose an algorithm other than the one it would have used if the adversary were passive, but the adversary cannot make a party use an unsupported algorithm. Moreover, assuming that the security of the building blocks is not breached before the confirmation messages are received, the client and the server are guaranteed to share the same transcript which includes negotiation messages. In other words, full downgrade resilience [7, 20] is satisfied once the other party is explicitly authenticated.

Comparison with KEMTLS. The assumption that the client knows the server public key from the onset of the protocol is precisely what allows to have the server send application data from its first message flow and to reduce the handshake by a full round-trip compared to the KEMTLS protocol. It also implies that the client need not verify the server certificate during the handshake, which speeds up the handshake even further and reduces power consumption.

However, as explained in the introduction, in a KEM-based protocol that achieves mutual authentication in a single round trip (see Figure 1), an adversary could a priori recover the client's identity by corrupting the long-term key of the server even after the handshake is completed (no forward identity protection), as it is for instance the case of the KEMTLS-PDK protocol [47]. The semi-static keys introduced in this paper mitigate this privacy loss and ensure, without extra round trip, that the client's identity cannot be recovered once the semi-static keys have expired. The lifetime of the semi-static keys now depends on

the desired trade-off between efficiency and privacy: the shorter the lifetime is, the stronger the privacy guarantees are for the client and the heavier the computational burden is on (mainly) the server.

Comparison with Session Resumption and Forward-Secret 0-RTT. TLS 1.3 specifies a session resumption (pre-shared key / "PSK") handshake, bootstrapping from symmetric secret keys that have been established in a prior connection and also enabling a 0-RTT mode. As also discussed in [47], the PSK handshake has efficiency advantages (e.g., for relying purely on symmetric cryptography) but also downsides wrt. key management of symmetric keys which need to be frequently changed (requiring additional communication) and stored securely in clients' memory. Our approach in contrast only stores (semi-static and long-term) *public* keys of the server at the client, reducing the risk for compromise as well as communication overhead.

The 0-RTT mode of TLS 1.3 enables clients to send application data in the first message flow, and thus reduce the handshake by a round trip compared to the standard mode. This requires servers to reconstruct secrets from previous sessions when receiving the clients' first messages, i.e., the 0-RTT mode is a resumption mechanism.

The standard resumption technique to achieve forward-secrecy and resilience to replay attacks consists in having servers store session caches (resumption secrets from all recent sessions) in local databases and issuing clients unique lookup keys that they use for their next connections. Similar techniques could a priori be used to reduce the KEMTLS handshake while maintaining forward identity protection, provided that the resumption handshake uses a KEM to achieve post-quantum security. The presented approach with semi-static keys in contrast obviates the need for extra secure updatable storage on the client side for resumption keys. It also allows the server to save storage by re-using $sk_s^{t_s}$ with many clients; session caches can easily grow huge.

Aviram, Gellert, and Jager [1, 2] proposed a different approach to forward secrecy based on puncturing techniques, improving over session caches in terms of server storage. Yet, at a 128-bit security level this easily requires tens of MB of server storage, compared to , e.g., a 2.342 kB single Kyber key pair with our protocol.

The main benefits of our protocol over forward-secret session resumption are therefore in small storage overhead (mainly on the server side), not needing (expensive) updatable secure client storage, and reliance on standardized post-quantum KEM components.

7 Implementation

This section discusses the implementation choices for the handshake protocol in Section 3. Since certificates are pre-distributed and need not be verified during handshakes, the main performance bottleneck depends on the choice of underlying KEMs. The main protocol is subsequently denoted PDK-SS (pre-distributed keys with semi-static contributions) for simpler referencing.

7.1 Choice of Primitives

The KEMs considered for implementation are among the finalists and alternates in the third round of the NIST Post-Quantum Standardization Process [41], with parameters chosen at security level 1 (roughly equivalent to the security of AES-128). The criteria of particular relevance in the IoT use case include the speed of cryptographic operations, the size of ciphertexts that may impact the handshake latency, and the size of the keys stored on devices and transmitted during the handshake.

We compare three of the NIST Round 3 finalist KEMs that rely on hardness assumptions over structured lattices and achieve good performance in terms of speed and size. These are Kyber512 [45] with security relying on the Module Learning with Errors (MLWE) problem, LightSABER [15] relying on the Module Learning with Rounding (MLWR) problem and NTRU-HPS-2048-509 [14] with security relying on the NTRU problem. We also include Round 3 Alternate candidate SIKE [30] which is based on supersingular isogeny Diffie–Hellman, using parameter set SIKEp434-compressed. Despite slower operations compared to its lattice-based counterparts, SIKE benefits from the smallest key and ciphertext sizes of remaining candidates in the NIST process.

To verify (client) certificates, we combined these KEMs with Dilithium-II [39] (with Kyber512 and LightSABER) and Falcon512 [42] (with NTRU) based on the similar assumptions. For the smallest size instantiation based on SIKE, we used Falcon, which has the smallest signatures of the Finalists.

7.2 Prototype Implementation

To experimentally evaluate PDK-SS, we implemented it by modifying the prototype implementation of the KEMTLS protocols [46, 47] based on Rustls [8], a TLS library written in Rust. The prototype integrates implementations of the post-quantum primitives from PQClean [31] and the Open Quantum Safe (OQS) library [50]. For all implementations we used AVX2-accelerated code.

The implementation is available under permissive licenses at `https://github.com/AbuLSim/1RTT-KEMTLS`.

8 Benchmarking

Table 2 compares the main protocol with other mutually authenticated handshake protocols, some of which also leverage cached leaf certificates. Even though these experiments were run on a powerful server and not on IoT devices, they clearly demonstrate the performance benefits of the main protocol.

8.1 Methodology

We compare PDK-SS to TLS with cached certificates [44] (both TLS 1.3 using X25519/RSA2048 and post-quantum variants), and to KEMTLS, with [47] and

Table 2: Average time in ms for mutually authenticated handshakes with cached leaf certificates.

Mutually authenticated		30.9 ms RTT, 1000 Mbps				195.5 ms RTT, 10 Mbps			
		Client send req.	Client recv. resp.	Server expl. auth.	Server recv CFIN.	Client send req.	Client recv. resp.	Server expl. auth.	Server recv CFIN.
kemtls	SIKE-c	196.8	228.0	**228.0**	165.9	697.0	893.3	**893.2**	500.9
	MLWE/MSIS	95.0	126.2	**126.2**	64.1	598.1	794.2	**794.2**	401.6
	NTRU	95.1	126.3	**126.2**	64.2	594.8	791.0	**790.9**	398.4
Cached TLS	TLS 1.3	68.8	100.3	**66.0**	38.2	399.2	596.6	**396.5**	204.6
	SIKE-c	103.0	134.8	**101.6**	72.8	431.7	630.5	**430.3**	238.1
	MLWE/MSIS	64.3	95.9	**63.7**	33.8	400.3	619.4	**399.7**	224.7
	NTRU	66.0	97.8	**64.6**	35.7	397.9	596.7	**396.5**	204.2
PDK	SIKE-c	130.6	161.7	**130.5**	99.7	466.6	662.7	**466.5**	269.3
	Kyber	63.3	94.4	**63.2**	32.3	400.5	596.5	**400.4**	200.6
	NTRU	63.3	94.5	**63.3**	32.4	396.7	592.7	**396.6**	198.8
	SABER	63.4	94.5	**63.3**	32.5	399.3	595.3	**399.2**	200.4
PDK-SS	SIKE-c	126.8	157.8	**126.7**	91.9	474.1	670.2	**474.0**	276.5
	Kyber	63.5	94.6	**63.4**	32.5	402.0	598.3	**401.9**	201.5
	NTRU	63.5	94.7	**63.5**	32.6	397.6	593.6	**397.5**	199.4
	SABER	63.6	94.7	**63.5**	32.7	401.5	597.7	**401.5**	201.1
PDK-SS async	SIKE-c	170.6	201.7	**170.6**	129.7	672.6	868.7	**672.5**	475.1
	Kyber	94.7	125.9	**94.7**	63.8	614.7	810.8	**614.7**	403.0
	NTRU	94.8	125.9	**94.7**	63.8	597.5	793.5	**597.5**	398.0
	SABER	94.9	126.0	**94.8**	63.9	604.0	800.0	**603.9**	401.1
PDK-SS update	SIKE-c	127.5	158.5	**127.4**	92.5	474.1	670.2	**474.0**	276.5
	Kyber	63.5	94.7	**63.5**	32.6	402.1	598.4	**402.0**	202.2
	NTRU	63.6	94.7	**63.5**	32.6	398.1	594.1	**398.1**	200.0
	SABER	63.7	94.8	**63.6**	32.7	401.5	597.7	**401.5**	201.7

without [46] pre-distributed keys (the former is denoted PDK in Table 2). Cached TLS is included for the sake of comparison to a real-world Internet protocol.

We analyze the performance of the PDK-SS protocol in three cases:

- the synchronized case PDK-SS, where the client and server share the same semi-static key;
- the asychronized case PDK-SS async, where the client and server have out-of-sync copies of the semi-static key and so the server must send its key to the client;
- the PDK-SS update case, where the client and server share the same semi-static server key but an update to the next semi-static epoch key is available.

The numbers in each column of Table 2 represent the average time to reach the corresponding stage of the protocol, measured in milliseconds over 60,000 handshakes for each scheme and each set of network parameters. The handshakes were performed on an emulated network; the experiment code is included in the source code repository. The server running the simulations was equipped with two Intel Xeon Gold 6230 CPUs, each with 20 cores. The left hand columns were computed over a low-latency, high-bandwidth (30.9 ms round

trip and 1000 Mbps) connection, with the right hand over a high-latency, low-bandwidth (195.5 ms round trip and 10 Mbps) connection. For each handshake, we measured the time taken for the client to send its first request in the form of application data, the client to receive the server response, the server to be explicitly authenticated, and finally the server to receive the client finished message. The time taken for the server to be explicitly authenticated is in bold font as we view it as the most important metric for our use case.

8.2 Analysis

Table 2 shows that the performances of PDK-SS (in the synchronized case), PDK and cached TLS are similar. That is because they are all 1-RTT, and the handshake time is dominated by the number of round trips since computation and transmission times are dwarfed by the network latency. The only exception is with SIKE as KEM, as its operations are an order of magnitude (milliseconds versus microseconds) slower than those of the other KEMs.

As for the asynchronized case, PDK-SS async compares most closely with the original KEMTLS handshake (PDK-SS async is somewhat faster as clients do not verify server certificates); their additional round trip clearly impacts the overall handshake time as expected. More precisely, PDK-SS is 44 to 49% faster than KEMTLS in the low-latency setup, and 46 to 49% faster in the high-latency setup.

Overall, our experiments confirm that the privacy benefits of introducing semi-static keys come at a negligible performance cost.

Acknowledgments. The authors thank Kenny Paterson and Cédric Fournet for helpful discussions. This work was supported by the Eurostars ZERO-TOUCH Project (E113920) and the European Research Council under Grant Agreement No. 805031 (EPOQUE). Felix Günther was supported in part by German Research Foundation (DFG) Research Fellowship grant GU 1859/1-1.

References

1. N. Aviram, K. Gellert, and T. Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 117–150. Springer, Heidelberg, May 2019.
2. N. Aviram, K. Gellert, and T. Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. *Journal of Cryptology*, 34(3):20, July 2021.
3. M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 602–619. Springer, Heidelberg, Aug. 2006.
4. M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg, Dec. 2001.

5. 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.
6. 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.
7. K. Bhargavan, C. Brzuska, C. Fournet, M. Green, M. Kohlweiss, and S. Zanella-Béguelin. Downgrade resilience in key-exchange protocols. In *2016 IEEE Symposium on Security and Privacy*, pages 506–525. IEEE Computer Society Press, May 2016.
8. J. Birr-Pixton. A modern TLS library in rust. `https://github.com/ctz/rustls`.
9. C. Boyd and K. Gellert. A modern view on forward security. Cryptology ePrint Archive, Report 2019/1362, 2019. `https://eprint.iacr.org/2019/1362`.
10. C. Brzuska. *On the Foundations of Key Exchange*. PhD thesis, Technische Universität, Darmstadt, 2013.
11. 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 2011*, pages 51–62. ACM Press, Oct. 2011.
12. 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.
13. 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. `https://eprint.iacr.org/2002/120/`.
14. C. Chen, O. Danba, J. Hoffstein, A. Hulsing, J. Rijneveld, J. M. Schanck, P. Schwabe, W. Whyte, Z. Zhang, T. Saito, T. Yamakawa, and K. Xagawa. NTRU. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.
15. J.-P. D'Anvers, A. Karmakar, S. S. Roy, F. Vercauteren, J. M. B. Mera, M. V. Beirendonck, and A. Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.
16. H. Davis and F. Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029, 2020. `https://eprint.iacr.org/2020/1029`.
17. D. Diemert and T. Jager. On the tight security of TLS 1.3: Theoretically-sound cryptographic parameters for real-world deployments. Cryptology ePrint Archive, Report 2020/726, 2020. `https://eprint.iacr.org/2020/726`.
18. 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 2015*, pages 1197–1210. ACM Press, Oct. 2015.
19. B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology*, 34(4):37, Oct. 2021.
20. B. Dowling and D. Stebila. Modelling ciphersuite and version negotiation in the TLS protocol. In E. Foo and D. Stebila, editors, *ACISP 15*, volume 9144 of *LNCS*, pages 270–288. Springer, Heidelberg, June / July 2015.
21. Smartm2m; guidelines for security, privacy and interoperability in iot system definition; a concrete approach. Technical Report ETSI SR 003 680, ETSI, 2020.

22. M. Fagan, K. Megas, K. Scarfone, and M. Smith. Foundational cybersecurity activities for iot device manufacturers. Technical Report NISTIR 8259, NIST, 2020.
23. M. Fagan, K. Megas, K. Scarfone, and M. Smith. Iot device cybersecurity capability core baseline. Technical Report NISTIR 8259A, NIST, 2020.
24. 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 2014*, pages 1193–1204. ACM Press, Nov. 2014.
25. M. Fischlin and F. Günther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017*, pages 60–75. IEEE, Apr. 2017.
26. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, Oct. 1984.
27. P. Grubbs, V. Maram, and K. G. Paterson. Anonymous, robust post-quantum public key encryption. Cryptology ePrint Archive, Report 2021/708, 2021. `https://eprint.iacr.org/2021/708`.
28. F. Günther. *Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols*. PhD thesis, Technische Universität, Darmstadt, 2018.
29. D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Y. Kalai and L. Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, Nov. 2017.
30. D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, D. Urbanik, G. Pereira, K. Karabina, and A. Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.
31. M. Kannwischer, J. Rijneveld, P. Schwabe, D. Stebila, and T. Wiggers. PQClean: Clean, portable, tested implementations of post quantum cryptography. `https://github.com/pqclean/pqclean`.
32. H. Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, Heidelberg, Aug. 2003.
33. 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.
34. H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978, 2015. `https://eprint.iacr.org/2015/978`.
35. H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. In *2016 IEEE European Symposium on Security and Privacy, EuroS&P 2016*, pages 81–96. IEEE, Mar. 2016.
36. K. Kwiatkowski and L. Valenta. The tls post-quantum experiment. https://blog.cloudflare.com/the-tls-post-quantum-experiment/, 2019.
37. B. A. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In W. Susilo, J. K. Liu, and Y. Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, Nov. 2007.
38. A. Langley. Cecpq2. https://www.imperialviolet.org/2018/12/12/cecpq2.html, 2018.
39. V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of

Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.
40. P. Mohassel. A closer look at anonymity and robustness in encryption schemes. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 501–518. Springer, Heidelberg, Dec. 2010.
41. NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. Technical report, 2016.
42. T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.
43. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), Aug. 2018.
44. S. Santesson and H. Tschofenig. Transport Layer Security (TLS) Cached Information Extension. RFC 7924, July 2016.
45. P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, and D. Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.
46. P. Schwabe, D. Stebila, and T. Wiggers. Post-quantum TLS without handshake signatures. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 1461–1480. ACM Press, Nov. 2020.
47. P. Schwabe, D. Stebila, and T. Wiggers. More efficient post-quantum KEMTLS with pre-distributed public keys. In *ESORICS 2021*, volume 12972 of *LNCS*, pages 3–22. Springer, Oct. 2021.
48. K. Sjöberg, P. Andres, T. Buburuzan, and A. Brakemeier. C-ITS deployment in europe - current status and outlook. *CoRR*, abs/1609.03876, 2016.
49. F. Song. A note on quantum security for post-quantum cryptography. In M. Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014*, pages 246–265. Springer, Heidelberg, Oct. 2014.
50. D. Stebila and M. Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In R. Avanzi and H. M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 14–37. Springer, Heidelberg, Aug. 2016.