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

Felix Günther and Patrick Towa

Department of Computer Science, ETH Zurich, Zurich, Switzerland

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 compared to TLS 1.3 in terms of latency is that KEMTLS introduces an additional round trip before the server can send data. 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. The 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 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.

1 Introduction

The Transport Layer Security (TLS) protocol is a widely deployed cryptographic protocol. It is used to securely access web pages, email servers, Internet-of-Things (IoT) gateways or even servers in Cooperative Intelligent Transport Systems [30] (C-ITSs). In the TLS *handshake* sub-protocol, a client and a server authenticate each other (at least the server to the client) and 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 [28], is TLS 1.3 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 [25, 27] 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 [32] 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 [29]. 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 [24] (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 settings or embedded devices, 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, as in the web setting, in practice is crucial to reduce network latency and power consumption.

Indeed, in the KEMTLS protocol, the client can only send application data after two round trips (i.e., it is a two-Round-Trip-Time or 2-RTT protocol) and the server cannot send application data before the client does, which contrasts with TLS 1.3 wherein the server can send application data (e.g., a server banner, an IoT-hub certificate) from its first message flow and the client can do so after a single round trip. The underlying reason is that after her initial message, the client must wait for the server public key to then encapsulate against it in order to implicitly authenticate the server, 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 a Single Round Trip. 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 issue is that the TLS protocol is expected to also provide *identity protection* [22], 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 she already knows to encrypt her certificate (as illustrated on Figure 1), but since there is no ephemeral contribution from the server yet, an adversary that compromises the server secret key could recover the client's identity even *after* the handshake is completed. In other words, there would be no *forward(-secure)* identity protection.

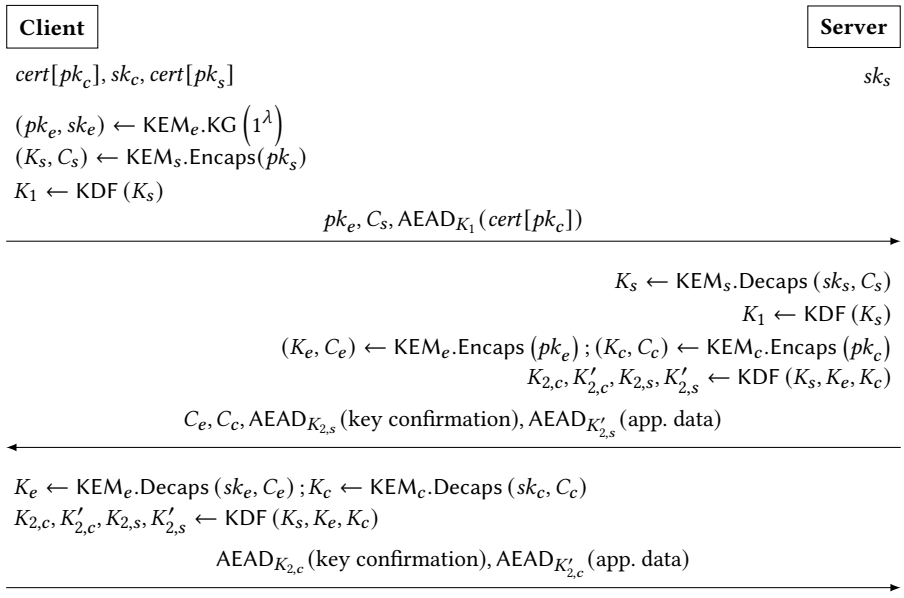

Fig. 1: A 1-RTT Protocol without Forward Identity Protection.

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 [15]. 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 the IoT device" [16] 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" [17]; in the present case, 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 first 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 on key encap-

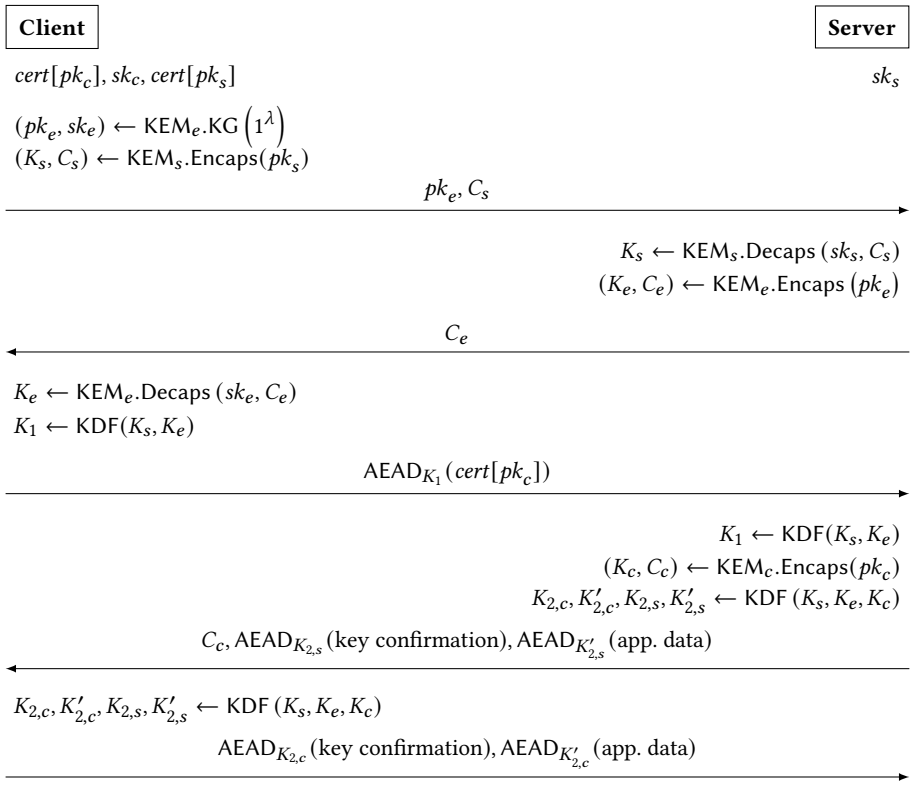

Fig. 2: A 2-RTT Protocol with Forward Identity Protection.

sulation, 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 know 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, and does so without any extra round compared to a 1-RTT protocol without forward identity protection. As 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 model for authenticated key exchange

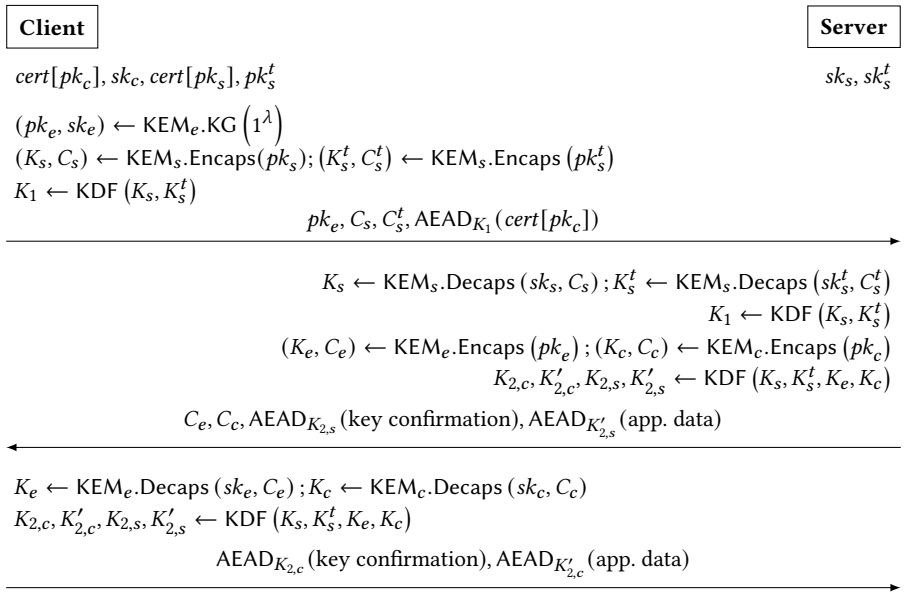

Fig. 3: Sketch of the Main Protocol.

proposed for TLS 1.3 by Dowling, Fischlin, Günther and Stebila [12, 13] and that for KEMTLS [29], 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.

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 $\{1, \ldots, n\}$.

2.2 Hash Functions

A hash function $H\colon \mathcal{X} \to \{0,1\}^\ell$ 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 Pseudo Random Function (PRF) [19] 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

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

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 [1] 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 [23] $\mathsf{KDF}(SKM, XTS, CTX, L) \to K$ takes as input a source key material SKM, an extractor-salt value XTS, some context information CTX and a length L, and returns an L-bit string K.

Hash-Based Key Derivation. Krawczyk proposed [23] a hash-bashed KDF that follows the extract-then-expand paradigm. It consists of two functionalities, namely an $\mathsf{Extract}(SKM, XTS) \to \mathrm{PRK}$ algorithm that computes a key for pseudo-random evaluation from the source key material and the salt, and an $\mathsf{Expand}(\mathrm{PRK}, CTX, L) \to K$ 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 $\mathsf{KG}\left(1^{\lambda}\right) \to K$ 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 against 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,

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

with $\mathcal{O}$ 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

A Key-Encapsulation Mechanism (KEM) is a public-key primitive which allows a party to privately send a symmetric key to another party using the public key of the latter. It consists of an algorithm $\mathsf{KG}\left(1^\lambda\right) \to (pk, sk)$ 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 [21] if

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

The security of a KEM requires the keys it generates to be indistinguishable from 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,

$$\left|\Pr\left[b = b' : \begin{array}{c}\left(pk^\star, sk^\star\right) \leftarrow \mathsf{KG}\left(1^\lambda\right) \\ b \leftarrow_{\$} \{0,1\} \\ \left(K_0^\star, C_0^\star\right) \leftarrow \mathsf{Encaps}\left(pk^\star\right) \\ K_1^\star \leftarrow_{\$} \mathcal{K} \\ b' \leftarrow \mathscr{A}^{\mathcal{O}_{atk}\left(sk^\star, C^\star, \cdot\right)}\left(pk^\star, C^\star, K_b^\star\right)\end{array}\right] - 1/2\right| \leq \varepsilon(\lambda),$$

with $\mathcal{O}_{atk}$ 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 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 application data after a single round trip, and the server from its first flow, as in TLS 1.3. In comparison, in the KEMTLS protocol [29] 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

Components. 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 [23] as keystone of the key schedule to extract randomness from the KEM-generated secrets and derive stage keys, and
- HMAC [2] as message-authentication code for explicit party authentication.
- 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 [24]. However, it strongly relies on an independent idea to reconcile client privacy (even if server long-term keys are later compromised) and a 1-RTT handshake: it introduces *server semi-static KEM keys* which the client encapsulates and mixes 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 she received from the server along with the corresponding time period, which is indicated by the server. It 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 she 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, and the client can encapsulate it and encrypt her certificate anew. This delays by a full round trip the step from which the client can send application data, but it is only for the first handshake during the time period. Afterwards, the client can do so after a single round trip.

Protocol Notation. Table 1 summarizes the notation used for the protocol secrets. Moreover, on 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 her 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.

Protocol Steps. The main steps of the protocol are as follows.

- The client first generates a pair (pk_e, sk_e) of ephemeral keys. She encapsulates against the server long-term and semi-static keys, and sends the resulting ciphertexts C_s resp. $C_s^{t_{s,c}}$ together with pk_e and a fresh nonce n_c in a `ClientHello` message. The latter message also contains the latest time period $t_{s,c}$ for which the client has a semi-static key from server s as well as the list of algorithms the client supports.
- The KEM secret K_s encapsulated in C_s is used to compute an early key-schedule secret ES, which implicitly authenticates the server as K_s cannot be efficiently recovered without sk_s.
- Next, the secret $K_s^{t_{s,c}}$ encapsulated in $C_s^{t_{s,c}}$ is injected into the key schedule to compute a handshake secret HS and further implicitly authenticate the server in period $t_{s,c}$.
- The client derives a handshake-traffic secret CHTS from the handshake secret and computes a key from it to AEAD-encrypt and send her certificate in a dedicated `ClientCertificate` message. This ensures that only a party with the knowledge of both the long-term and the semi-static secret key used can infer the client's identity.

– When the server receives the client's initial `ClientHello` and `ClientCertificate` messages, two cases arise: either the client time period $t_{s,c}$ matches the current server time period t_s or not.

  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 CHTS and decrypt the client certificate. Both parties compute a derived handshake secret dHS that they keep as current state of the key schedule.

  * The server encapsulates against pk_c and pk_e and sends in a `ServerHello` message the corresponding ciphertexts C_c and C_e, together with a fresh nonce and the algorithms selected from the algorithms supported by the client.
  * The two parties now compute an intermediate master secret from the KEM secret K_e encapsulated in C_e, then expand a derived intermediate master secret dIMS and finally compute a master secret MS from the KEM secret K_c encapsulated in C_c. Secret K_e enables forward secret and K_c implicitly authenticates the client.
  * From the master secret, both parties compute (mutually) authenticated handshake-traffic secrets SAHTS and CAHTS for the server and the client. These are used to derive AEAD keys to encrypt the rest of the handshake messages.
  * Both parties compute from the master secret MAC "finished" keys fk_s and fk_c for explicit authentication as well application-transport secrets SATS and CATS from which AEAD keys are derived to encrypt application data.
  * The server explicitly authenticates himself by computing a MAC of the transcript with fk_s, i.e., a confirmation message, and can now send application data.

    During the transition phase from the current time period to the next, the server also sends the public key pk_s^{t+1} for the next time period.[1] The client saves this key only after verifying the server confirmation message.
  * Upon receiving the server "finished" message, the client explicitly authenticates herself to the server by computing a MAC of the transcript with fk_c and can send application data.

  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 on Figure 5, the server does not hold $sk_s^{t_{s,c}}$ and cannot compute the client handshake-traffic secret (denoted CHTS′) and recover the client certificate. The server therefore rejects the stage-1 key.

  * Yet, the server can already encapsulate the ephemeral key and inject it into the key schedule to achieve forward secrecy for the following stage keys.
  * Both parties can now compute server and client handshake-traffic secrets SHTS and CHTS, and derive keys to encrypt handshake messages.
  * Since the client does not have the current server semi-static key $pk_s^{t_s}$, the server sends it to the client for the following handshakes in period t_s, though it is not useful in this handshake anymore to protect the client certificate: the

[1] The server does so once per client that follows the protocol, as such a client switches to the next key for the subsequent handshakes and as the server already accepts the next key during the transition phase from the current period to the next.

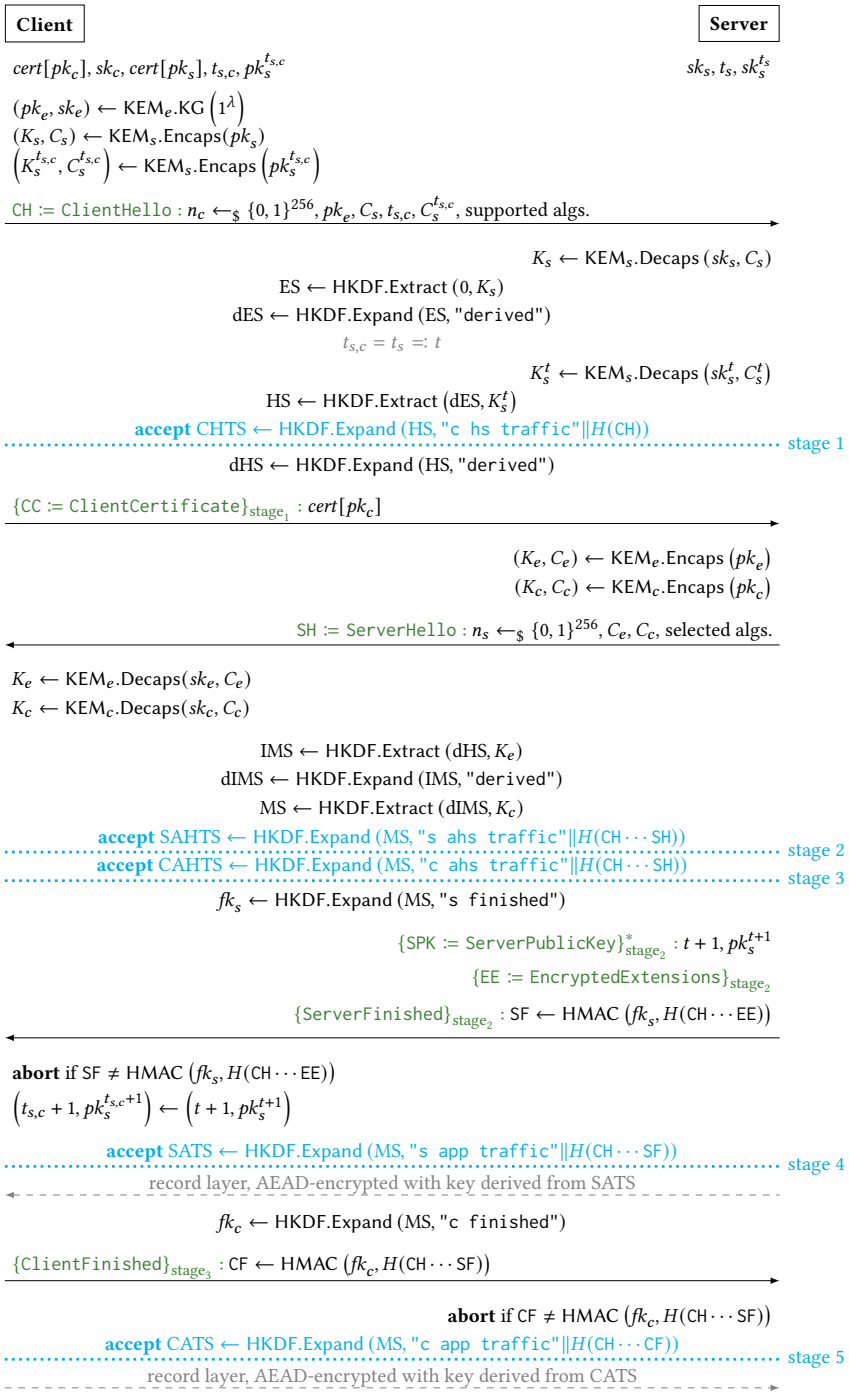

Fig. 4: Protocol in the Case of Matching Time Periods.

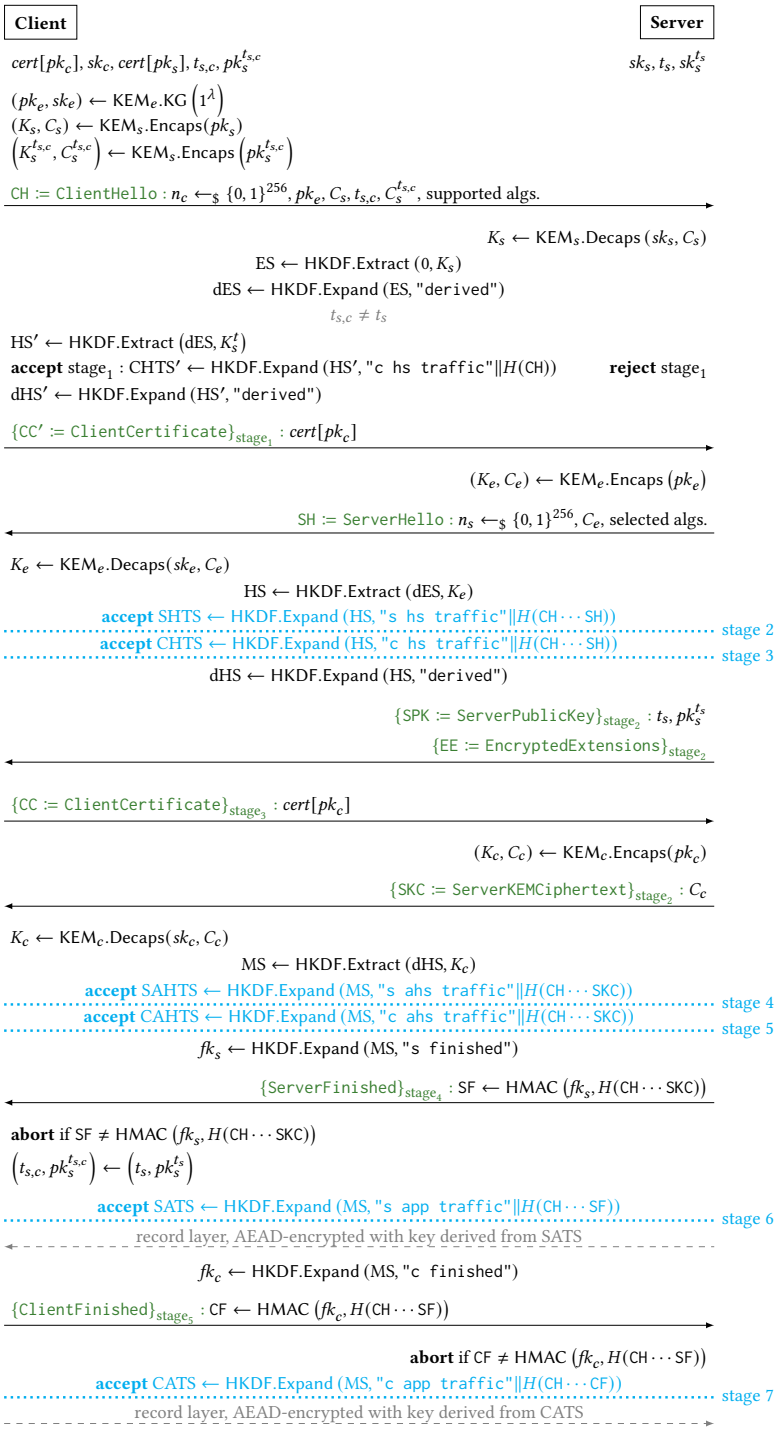

Fig. 5: Protocol in the Case of Unmatching Time Periods.

ephemeral secret has already been mixed into the key schedule and the client certificate is encrypted with CHTS which is fully forward secret. Once again, the client only saves this semi-static key after verifying the server confirmation message.

* The client sends her certificate and in this case benefits from better privacy than in the case $t_{s,c} = t_s$ since the certificate is protected by a fully ephemeral secret rather than a semi-static one.
* The server encapsulates the client public key and implicitly authenticates her by using the resulting KEM secret to compute a master secret.
* The parties proceed with confirmation MAC values as in the case $t_{s,c} = t_s$. □

Discussion. In case an adversary were to change $t_{s,c}$, the parties would indeed 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, meaning that 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 these are 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.

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 Protocol Secrets.

Application to the KEMTLS Protocol 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 protocol with client authentication [29, 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 Model

This section introduces the model for 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 [12, 13] and that for KEMTLS by Schwabe, Stebila, and Wigger [29]. Their models follow a line of work [18, 20] 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 [3] that introduced the paradigm of session-key indistinguishability, and for which Brzuska et al. [6,7] 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 the long-term and semi-static secrets of honest parties and the keys established during protocol runs. The protocol is then deemed secure if such an adversary cannot distinguish a key established at a stage of a non-compromised 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, i.e., a stage key can be *retroactively* explicitly authenticated.

Forward Secrecy. The model also 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 or not.

- A stage key satisfies *(full) forward secrecy* if the adversary either 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. In this sense, this (informal) definition of delayed forward secrecy is related to Boyd and Gellert's [5].

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 (to for example encipher application message with a symmetric scheme).

4.1 Syntax

Similarly to the model for TLS 1.3 proposed by Dowling et al. [12, 13], 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[2]. 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.

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 p. 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 $id \in \Sigma$[3]and the identity of the intended partner client is currently unknown, the special 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 [9].
- $role \in \{initiator, responder\}$: specifies 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 once (and 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. [7], 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

[3] Post-specified peers are only allowed in the case of 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.

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.

Moreover, in addition to the variables maintained by each session run by the challenger, the latter maintains for each session σ that it runs, the following game variables.

- $period \in \mathbb{N} \cup \{-\infty\}$: 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 uniformly random key. All entries are initially set to FALSE.

Game Oracles. In the security games, the adversary is given access to the following oracles.

- $\mathsf{NewSession}(p, q, role)$: 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 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 $\left(\sigma.stage, \sigma.status_{\sigma.stage}\right)$. 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 σ' such that $\sigma.sid_{\sigma.stage} = \sigma'.sid_{\sigma.stage}$ and $\sigma'.tested_{\sigma.stage} = \text{TRUE}$, then
  - ▹ the challenger sets $\sigma.tested_{\sigma.stage} \leftarrow \text{TRUE}$. 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 $\sigma.revealed_i \leftarrow \text{TRUE}$.
– $\mathsf{NextPeriod}(s \in \Sigma)$: 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 $\sigma'.tested_i = \text{TRUE}$, 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 $\sigma.tested_i \leftarrow \text{TRUE}$.
  * $K \leftarrow \sigma.key_i$ if $b = 0$, otherwise K is drawn uniformly at random from the key space.
  * If $\sigma.use_i = internal$, then 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 $\sigma'.tested_i \leftarrow \text{TRUE}$, 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 [8, 26]. 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* $(M, FS, iauth_s, iauth_c, eauth, use)$*, 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* $G^{\text{Match}}_{\mathsf{KE},\mathscr{A}}$*.*

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* (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 specified above (b parametrizes the test oracle).*

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

$G^{\text{Match}}_{\mathsf{KE},\mathscr{A}}$ *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 stage, i.e., three pairwise distinct sessions* σ*,* σ' *and* σ'' *and a stage* $i \in [\![M]\!]$ *such that* $\sigma.sid_i = \sigma'.sid_i = \sigma''.sid_i \neq \perp$*.*
2. *Two sessions share the same identifier at some stage but have non-opposite roles, i.e., two distinct sessions* σ *and* σ' *and a stage* $i \in [\![M]\!]$ *such that* $\sigma.sid_i = \sigma'.sid \neq \perp$ *and* $\sigma.role = \sigma'.role$*.*
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 $\Pr\left[G^{\text{Match}}_{\text{KE},\mathscr{A}} \rightarrow 1\right]$ *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 $\sigma.sid_i = \sigma'.sid_i$, $\sigma'.revealed_i =$ FALSE.*
2. *If $\sigma.FS_i = fs$, then*
   (a) *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 to say, $\sigma.status_{\sigma.eauth_{i,1}} =$ 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. *there exists a session $\sigma' \neq \sigma$ such that $\sigma.cid_i = \sigma'.cid_i$ and $\sigma.role \neq \sigma'.role$, and $sk^{\sigma.period}_{\sigma.id}$ 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 $sk^{\sigma.period}_{\sigma.id}$ is not corrupt.*

On Freshness. Beyond ruling out the trivial attack of both revealing and testing a session key (within the same or two partnered sessions), 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 ephemeral values are involved in their computation. The closest equivalent in this case are semi-static values. 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 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., $\sigma.eauth_{i,1}$ if $\sigma.role = initiator$ and $\sigma.eauth_{i,2}$ 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 $(M, FS, iauth_s, iauth_c, eauth, use)$, 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.
- *If a tested stage is not fresh then $b' \leftarrow_\$ \{0,1\}$.*
- *If a session maliciously a stage then $b' \leftarrow b$.*

$G^{\text{Multi–Stage}}_{\mathsf{KE},\mathscr{A}}$ returns 1 if $b = b'$ and otherwise returns 0.

Protocol KE *satisfies* $(T, q_{\mathcal{O}}, \varepsilon)$*-multi-stage 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 $\mathcal{A}$ *that runs in time at most* $T(\lambda)$ *and makes at most* $q_{\mathcal{O}}(\lambda)$ *queries, the advantage* $\left|\Pr\left[G^{\text{Multi-Stage}}_{\mathsf{KE},\mathcal{A}} \rightarrow 1\right] - 1/2\right|$ *of* $\mathcal{A}$ *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 not been used in TLS-related models but appeared in models for other protocols [18]. The model handles these via the time-related identity and session game variables, the oracles that gives 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. [12, 13] 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 [29]), in contrast to the model from [12, 13] 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 [12,13]. This simplification is possible because both parties always authenticate themselves in the protocol from Section 3. Besides, the model in [12, 13] 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. Message replays, which are supported in 0-RTT handshakes based on pre-shared keys, are consequently not considered either.

Another major technical difference from the model by Dowling et al. [12, 13] 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 [29] 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 [12, 13].

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.

- $M = 5$ stages as shown in Figure 4.
- $FS = (dfs, fs, fs, fs, fs)$. That is, the first stage key satisfies delayed forward secrecy and all the others full forward secrecy.
- $iauth_s = 1$, the server is implicitly authenticated from stage 1.
- $iauth_c = 2$, the client is implicitly authenticated from stage 2.
- $eauth = ((4,5),(4,5),(4,5),(4,5),(5,5))$. The server is explicitly authenticated from stage 4 and the client from stage 5.
- $use = (internal : \{1,2,3\}, external : \{4,5\})$ The first three stage keys are used to encrypt handshake traffic, i.e., for internal use.

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. It the present case, denoting by $\mathsf{SC} := \texttt{ServerCertificate}$ the server certificate $cert[pk_s]$, the session identifiers are

$$\begin{aligned} sid_1 &= (\text{“CHTS”}, \mathsf{SC}, \mathsf{CH}), \\ sid_2 &= (\text{“SAHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{CC}, \mathsf{SH}), \\ sid_3 &= (\text{“CAHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{CC}, \mathsf{SH}), \\ sid_4 &= (\text{“SATS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{CC}, \mathsf{SH}, \mathsf{SPK}^*, \mathsf{EE}, \mathsf{SF}), \\ sid_5 &= (\text{“CATS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{CC}, \mathsf{SH}, \mathsf{SPK}^*, \mathsf{EE}, \mathsf{SF}, \mathsf{CF}). \end{aligned}$$

As for the contributive identifiers, the parties compute them as specified below.

- Upon sending (resp. receiving) the `ClientHello` message, the client (resp. the server) sets $cid_1 \leftarrow$ (“CHTS”, SC, CH).
- Upon sending (resp. receiving) the `ClientCertificate` message, the client (resp. the server) sets $cid_2 \leftarrow$ (“SAHTS”, SC, CH, CC).
- Upon sending (resp. receiving) the `ServerHello` message, the server (resp. the client) sets $cid_2 \leftarrow$ (“SAHTS”, SC, CH, CC, SH).
- The client and the server set

$$\begin{aligned} cid_3 &\leftarrow (\text{“CAHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{CC}, \mathsf{SH}), \\ cid_4 &\leftarrow (\text{“SATS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{CC}, \mathsf{SH}) \text{ and} \\ cid_5 &\leftarrow (\text{“CATS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{CC}, \mathsf{SH}) \end{aligned}$$

after they respectively compute $sid_3, \ldots, sid_5$.

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

- $M = 7$ stages as on Figure 5.
- $FS = (dfs, fs, fs, fs, fs, fs, fs)$. The first stage key, only accepted by the client, satisfies delayed forward secrecy and all the others full forward secrecy.
- $iauth_s = 1$. The server is implicitly authenticated from the start of the protocol.
- $iauth_c = 4$. The client secret key is mixed into the key schedule at stage 4.
- $eauth = ((\infty, \infty), (6, 7), (6, 7), (6, 7), (6, 7), (6, 7), (7, 7))$. The server is explicitly authenticated from stage 6, and the client from stage 7.
- $use = (internal : \{1, 2, 3, 4, 5\}, external : \{6, 7\})$ Only the application-transport keys which are computed at stages 6 and 7 are for external use.

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

$$
\begin{aligned}
sid_1 &= \begin{cases} (\text{“CHTS”}, \mathsf{SC}, \mathsf{CH}) & \text{if } \sigma.role = initiator \\ \perp \text{ otherwise,} \end{cases} \\
sid_2 &= (\text{“SHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}), \\
sid_3 &= (\text{“CHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}), \\
sid_4 &= (\text{“SAHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}, \mathsf{SPK}, \mathsf{EE}, \mathsf{CC}, \mathsf{SKC}), \\
sid_5 &= (\text{“CAHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}, \mathsf{SPK}, \mathsf{EE}, \mathsf{CC}, \mathsf{SKC}), \\
sid_6 &= (\text{“SATS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}, \mathsf{SPK}, \mathsf{EE}, \mathsf{CC}, \mathsf{SKC}, \mathsf{SF}), \\
sid_7 &= (\text{“CATS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}, \mathsf{SPK}, \mathsf{EE}, \mathsf{CC}, \mathsf{SKC}, \mathsf{SF}, \mathsf{CF}).
\end{aligned}
$$

To compute the contributive identifiers, the parties proceed as follows.

- Upon sending (resp. receiving) the `ClientHello` message, the client (resp. the server) sets $cid_1 \leftarrow (\text{“CHTS”}, \mathsf{SC}, \mathsf{CH})$.
- Upon sending the `ClientCertificate` message, the client sets $cid_2 \leftarrow (\text{“SHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{CC}')$.
- Upon receiving the `ClientHello` message, since the time periods do not match, the server sets $cid_1 \leftarrow sid_1 \leftarrow \perp$.
- Upon sending the `ServerHello` message, the server sets $cid_2 \leftarrow (\text{“SHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH})$.
- Upon receiving the `ServerHello` message, as it does not contain a KEM_c ciphertext, the client knows the time periods did not match and then updates $cid_2 \leftarrow (\text{“SHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH})$.
- The client and the server set

$$
\begin{aligned}
cid_3 &\leftarrow (\text{“CHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}), \\
cid_4 &\leftarrow (\text{“SAHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}), \\
cid_5 &\leftarrow (\text{“CAHTS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}), \\
cid_6 &\leftarrow (\text{“SATS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH}) \text{ and} \\
cid_7 &\leftarrow (\text{“CATS”}, \mathsf{SC}, \mathsf{CH}, \mathsf{SH})
\end{aligned}
$$

after they respectively compute $sid_3, \ldots, sid_7$.

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 if is enough for the SH to be delivered to prove the secrecy of *fs* stage keys (cf. Case (2) (c) in Definition 4.2), and also of *dfs* 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. 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, and it suffices to bound the probability that each of the winning conditions is satisfied.

1. *More than two sessions share the same identifier at some stage.* At each stage, the session identifier includes the random nonce from the `ClientHello` and `ServerHello` messages, so three pairwise distinct sessions can share the same identifier only if at least two of them share the same nonce. 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.* Assuming that at most two 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.
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 equal to the session identifier once the session accepts the stage.

5. *Two distinct stages share the same session identifier.* This event never occurs 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 formalize the multi-stage security of the main protocol.

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

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

S *is* $\left(T_{\mathsf{S}}^{\mathsf{A}}, q_{\mathsf{S}}^{\mathsf{A}}, \varepsilon_{\mathsf{S}}^{\mathsf{A}}\right)$*-*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*

$$\begin{aligned}\mathscr{O} \in \{&\mathsf{NewSession}, \mathsf{Send}, \mathsf{Reveal}, \mathsf{NextPeriod}, \mathsf{EndCurrentPeriod},\\ &\mathsf{Corrupt}, \mathsf{SemiStaticCorrupt}, \mathsf{Test}\}.\end{aligned}$$

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

$$2^{-257} n_\sigma^2 + \varepsilon_H^{\text{Coll}} + 5n_\sigma \left(\begin{array}{l} n_{id} \left(\begin{array}{l} \varepsilon_{\mathsf{KEM}_c}^{\text{IND-CCA}} + \varepsilon_{\mathsf{KEM}_s}^{\text{IND-CCA}} \\ + 3\varepsilon_{\mathsf{HKDF.Extract}}^{\text{PRF}} + \varepsilon_{\mathsf{HKDF.Extract}}^{\text{dual-PRF}} \\ + 5\varepsilon_{\mathsf{HKDF.Expand}}^{\text{PRF}} + \varepsilon_{\mathsf{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \\ + n_\sigma \left(\begin{array}{l} \varepsilon_{\mathsf{KEM}_e}^{\text{IND-1CCA}} + \varepsilon_{\mathsf{KEM}_s}^{\text{IND-CCA}} \\ + 2\varepsilon_{\mathsf{HKDF.Extract}}^{\text{PRF}} + \varepsilon_{\mathsf{HKDF.Extract}}^{\text{dual-PRF}} \\ + 4\varepsilon_{\mathsf{HKDF.Expand}}^{\text{PRF}} + \varepsilon_{\mathsf{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \end{array} \right),$$

with $n_{id} := |ID|$ *and* $\varepsilon_H^{\text{Coll}}$ *the probability that an algorithm given in the proof finds a collision for* H *by running* $\mathscr{A}$ *as subroutine.*

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 exists two distinct honest sessions with the same role that share 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.

The next step consists in restricting the adversary to a single Test query. 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 $t \in [\![5n_\sigma]\!]$ 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 [13, Appendix A] that (in their model,) the advantage of $\mathscr{B}$ is at least a fraction $5n_\sigma$ 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 $T_{\mathscr{A}} + O\left(q_{\mathsf{Send}} + q_{\mathsf{Test}}\right)$. 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 3. Only the first Test query in this game may be replied to with anything else than $\perp$. As explained above, there exists an algorithm that runs the adversary as subroutine and has an advantage of at least a $5n_\sigma$ fraction of the advantage of $\mathscr{A}$ in the previous game by making at most one Test query, $q_{\mathsf{Reveal}} + M_i\left(q_{\mathsf{Send}} + q_{\mathsf{Test}}\right)$ Reveal queries and the same amount of queries as $\mathscr{A}$ for the other oracles. Besides, the algorithm runs in time $T_{\mathscr{A}} + O\left(q_{\mathsf{Send}} + q_{\mathsf{Test}}\right)$.

For the unique Test query that the adversary may make on some session and stage (σ, i), which is now known in advance, two main cases are now distinguished:

A. $i = 1$, i.e., the adversary tests the key computed at stage 1 which satisfies delayed forward secrecy, and
B. $i > 1$, i.e., the adversary tests a stage key that satisfies forward secrecy.

These cases are further subdivided into the following sub-cases.

A.I. (a) $\sigma.role = initiator$ and $sk_{\sigma.pid}$ is not corrupt (recall that the server is implicitly authenticated from stage 1).
(b) $\sigma.role = initiator$ and $sk_{\sigma.pid}^{\sigma.period}$ is not corrupt.
A.II. (a) $\sigma.role = responder$ and there exists a session $\sigma' \neq \sigma$ such that $\sigma.cid_1 = \sigma'.cid_1$ and $\sigma.role \neq \sigma'.role$, and $sk_{\sigma.id}^{\sigma.period}$ is not corrupt.
(b) $\sigma.role = responder$, $\sigma.status_5 = accepted$ and the adversary did not corrupt $sk_{\sigma.pid}$ before σ accepted stage 5.

B.I. There exists a session $\sigma' \neq \sigma$ such that $\sigma.cid_i = \sigma'.cid_i$ and $\sigma.role \neq \sigma'.role$.
B.II. (a) $\sigma.role = initiator$ and $sk_{\sigma.pid}$ is not corrupt (recall that the server is implicitly authenticated from stage 1).
(b) $\sigma.role = responder$ and $sk_{\sigma.pid}$ is not corrupt (recall that $i \geq 2 = \sigma.iauth_c$ in case B).
B.III. (a) $\sigma.role = initiator$, $\sigma.status_4 = accepted$ and the adversary did not corrupt $sk_{\sigma.pid}$ before σ accepted stage 4.
(b) $\sigma.role = responder$, $\sigma.status_5 = accepted$, the adversary did not corrupt $sk_{\sigma.pid}$ before σ accepted stage 5 and $sk_{\sigma.pid}^{\sigma.period}$ is not corrupt.

Note that these cases somewhat correspond to the conditions for the tested session to be fresh, cf. Definition 4.2.

Case A.I.a: Tested Client with Non-Compromised Partner Long-Term Key

In this case, the secrecy of the client stage key relies on the indistinguishability of the KEM_c ciphertext C_c since the server is implicitly authenticated from stage 1.

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

Game A.I.a.1 (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.

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 public key of $\sigma.pid$. Then, it sets $C^\star$ as the ciphertext C_s in the `ClientHello` message of the test 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.2 (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 sent by σ.

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 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, and forward the decision bit of the adversary.

Game A.I.a.3 (dES**).** The challenger replaces the derived early secret dES in σ with a uniformly random value. The same value is used in any session of $\sigma.pid$ that received the ciphertext C_s sent by σ. The problem of distinguishing this game from the previous can be reduced to the PRF security of HKDF.Expand.

Game A.I.a.4 (HS**).** The handshake secret HS 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 sent by σ. Now the PRF security of HKDF.Extract serves as argument for the indistinguishability from the previous game.

Game A.I.a.5 (CHTS**).** The challenger of this game replaces client handshake traffic secret CHTS in σ with a uniformly random value. Moreover, in any session of $\sigma.pid$ that received the ciphertext C_s sent by σ, secret CHTS is also replaced with a uniformly random value. If the `ClientHello` message was not altered, then it is the same value as the one used in σ, otherwise it is an independent random value (recall that the game halts if the hashes of two distinct values collide). The PRF security of HKDF.Expand is now used to argue for the indistinguishability of this game from the previous.

Game A.I.a.6 (dHS**).** The derived handshake secret is here replaced with a uniformly random value. The same value is used in any session of $\sigma.pid$ that received the ciphertext C_s sent by σ. The indistinguishability of this game from the previous one follows from the PRF security of HKDF.Expand.

Game A.I.a.7 (IMS**).** In this game, the intermediate master secret IMS is 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 sent by σ. The PRF security of HKDF.Extract allows to argue for the indistinguishability between this game and the previous one.

Game A.I.a.8 (dIMS**).** This value is replaced with a uniformly value in σ and the same value is used in any session of $\sigma.pid$ that received the ciphertext C_s sent by σ. The indistinguishabilty from the previous game relies again on the PRF security of HKDF.Expand.

Game A.I.a.9 (MS**).** The master secret is replaced with a uniformly random value in σ and the same value is used in any session of $\sigma.pid$ that received the ciphertext C_s sent by σ. 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.10 (SAHTS**,** CAHTS**,** fk_s**,** SATS**,** fk_c **and** CATS**).** These values are replaced with uniformly random ones in σ and the same values are used in any session of $\sigma.pid$ that received the ciphertext C_s sent by σ. The PRF security of HKDF.Expand implies the indistinguishability between this game and the previous.

Game A.I.a.11 (MAC Forgery). The challenger of this game, running the tested client session, rejects the `ServerFinished` message in case there is no partner session at stage 4. The fact that there is no partner session at stage 4 implies that no honest session computed a MAC tag on the transcript of the tested client 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.

Note that since the challenger rejects all `ServerFinished` message in case there is no partner at stage 4, the event in which the client session accepts stage 4 without a partner session never occurs. Besides, all stage keys are uniformly random. The advantage of the adversary in the last game is therefore nil.

Besides, as a client session sets $pk'_{\sigma.pid,\sigma.id}$ received in a SPK message (during the transition from $\sigma.period$ to $\sigma.period + 1$, i.e., when $pk''_{\sigma.pid} \neq \perp$) only after verifying the tag of the `ServerFinished` message, the public semi-static key necessarily comes from the partner session in the last game.

It follows that the advantage of the adversary in Game 3 is in this case at most

$$n_{id}\left(\varepsilon^{\text{IND-CCA}}_{\text{KEM}_s} + 3\varepsilon^{\text{PRF}}_{\text{HKDF.Extract}} + \varepsilon^{\text{dual-PRF}}_{\text{HKDF.Extract}} + 5\varepsilon^{\text{PRF}}_{\text{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\text{HMAC}}\right).$$

Case A.I.b: Tested Client with Non-Compromised Partner Semi-Static Key

This case is fairly similar to the previous one, except that the secrecy of the stage-1 key now relies on the indistinguishability of the semi-static ciphertext $C_s^{t_{s,c}}$.

Game A.I.b.0 (Guess Partner Identity). Same as Game A.I.a.0.

Game A.I.b.1 (Semi-Static KEM). Similar to Game A.I.a.1, but with $C_s^{t_{s,c}}$ and $K_s^{t_{s,c}}$ instead of C_s and K_S. Note that the reduction algorithm must set the challenge KEM public key $pk^\star$ as the semis-static key of $\sigma.pid$ in $\sigma.period$.

Game A.I.b.2 (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 $C_s^{t_{s,c}}$ sent by σ, the handshake secret HS is also replaced with a uniformly random value, while maintaining 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 the session dES ($K_s^{t_{s,c}}$ 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 $C_s^{t_{s,c}}$ 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 $C_s^{t_{s,c}}$ 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.

Games A.I.b.3–A.I.b.9. Identical to Games A.I.a.5 to A.I.a.11.

In this case, the advantage of the adversary in Game 3 is therefore at most

$$n_{id}\left(\varepsilon^{\text{IND-CCA}}_{\text{KEM}_s} + 2\varepsilon^{\text{PRF}}_{\text{HKDF.Extract}} + \varepsilon^{\text{dual-PRF}}_{\text{HKDF.Extract}} + 4\varepsilon^{\text{PRF}}_{\text{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\text{HMAC}}\right).$$

Case A.II.a: Tested Server with Honest Contributive Partner

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 no two sessions with the same role can share the same nonce, there is exactly one session that shares the same contributive identifier at stage 1. Denote this session by σ'.

Game A.II.a.0 (Guess Contributive Session). The challenger guesses at the beginning of the game the contributive partner session and aborts and returns 0 if the guess is incorrect when the adversary makes its test query. This decreases the advantage of the adversary by a factor at most n_σ.

Game A.II.a.1 (Semi-Static KEM). In this game, the challenger replaces the semi-static value $K_s^{t_{s,c}}$ in both σ and σ' with a uniformly random value (the same).
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 $\sigma.cid_1 = \sigma'.cid_1$, 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'. It uses its decapsulation oracle to decapsulate any other ciphertexts for the public key $pk^\star$ of $\sigma.id$ in $\sigma.period$. The reduction algorithm eventually forwards the decision bit of the adversary to the IND-CCA challenger.

Games A.II.a.2–A.II.a.8. Similar to Games A.I.a.4 to A.I.a.10, except that the replacement is now only for σ and σ'. This implies that the reduction algorithms need only make one oracle query in each of the security games for HKDF.Extract and HKDF.Expand.

Game A.II.a.9 (MAC Forgery). The challenger of this game, running the tested server session, rejects the `ClientFinished` message in case there is no partner session at stage 5. The fact that there is no partner session at stage 5 implies that no honest

session computed a MAC tag on the transcript of the tested client 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 3 is in this case at most

$$n_\sigma \left(\varepsilon^{\text{IND-CCA}}_{\text{KEM}_s} + 2\varepsilon^{\text{PRF}}_{\text{HKDF.Extract}} + \varepsilon^{\text{dual-PRF}}_{\text{HKDF.Extract}} + 4\varepsilon^{\text{PRF}}_{\text{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\text{HMAC}} \right).$$

Case A.II.b: Tested Server with Explicitly Authenticated Client and Non-Compromised Semi-Static Key

The client's secret key not being compromised before acceptance of stage 5 allows to distinguish two cases: either the tested session accepts at stage 5 with an honest partner, in which case the conditions for the prior Case A.II.a are satisfied; or the tested 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 test session accept, is secret. The secrecy of the MAC key is established as in Case A.I.a, but is based on the client long-term KEM key K_c, the subsequent sequence of key derivations and the unforgeability of the MAC:

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

Case B.I: Honest Contributive Partner

For keys with expected forward secrecy, the existence of an honest contributive partner 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 is exactly one contributive partner. Denote it by σ'. Let $\sigma_c \in \{\sigma, \sigma'\}$ be such that $\sigma_c.role = initiator$ and $\sigma_s \in \{\sigma, \sigma'\}$ such that $\sigma_s.role = responder$.

Game B.I.0 (Guess Contributive Session). Same as Game A.II.0.

Game B.I.1 (Ephemeral KEM) The challenger of this game replaces the ephemeral secret value K_e in σ_s with uniformly random value, and uses the same value in σ_c if the latter receives the ciphertext σ_s sent.

Distinguishing this game from the previous one can be reduced to the IND-1CCA security of KEM_e as follows. Upon receiving the challenger tuple $(pk^\star, C^\star, K^\star)$, the reduction algorithm sends $pk^\star$ as the ephemeral public key in the `ClientHello` message. Note that since $\sigma_c.cid_1 = \sigma_s.cid_1$, this ephemeral public key is delivered to σ_s. The reduction algorithm then uses $C^\star$ as the ephemeral ciphertext in the `ServerHello` message and $K^\star$ as the ephemeral secret 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 decapsulation query to the IND-1CCA challenger 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.2 (IMS**).** In this game, the intermediate master secret MS in σ_s is replaced with a uniformly random value, and the same value is used in σ_c if the latter received the ephemeral ciphertext sent by σ_s. The indistinguishability of this game from the previous can be reduced to the dual PRF security of HKDF.Extract. The reduction algorithm simply has a to make a single oracle query on dHS (the key of the dual PRF challenger is implicitly set as K_e).

Game B.I.3 (dIMS**).** The challenger of this game replaces the derived intermediate master secret dIMS in σ_s with a uniformly random value. If σ_c received the ciphertext sent by σ_s, then dIMS is also replaced in σ_c with the same value (note that since $\sigma_c.cid_1 = \sigma_s.cid_1$, then the dES value is the same in both sessions). The PRF security of HKDF.Expand (with a single oracle query) substantiates the indistinguishability of this game from the previous one.

Game B.I.4 (MS**).** The master secret MS in σ_s is here replaced with a uniformly random value. If σ_c received the ciphertext sent by σ_s, then the same value is used in σ_c. The PRF security of HKDF.Extract (with a single oracle query) justifies the indistinguishability from the previous game.

Game B.I.5 (SAHTS**,** CAHTS**,** fk_s**,** SATS**,** fk_c **and** CATS**).** These values are now replaced with uniformly random ones in σ_s. The same values are used in session σ_c if σ_c and σ_s are partnered at stage i (which implies that both sessions have accepted stage i), and they are otherwise replaced with independent uniformly random values. Note that if the two sessions 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. The indistinguishability from the previous game follows from the PRF security of HKDF.Expand.

Game B.I.6 (MAC Forgery). The challenger of this game rejects the finished message of the intended partner message in case there is no partner session at the stage explicit partner authentication (4 if $\sigma.role = initiator$ and 5 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 tested client 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 3 is thus in this case at most

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

Case B.II.a: Tested 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 very similarly to case A.I.a, except that there is no need to replaced CHTS with a uniformly random value as it is not tested and it is not involved in the computation of further values.

Games B.II.a.0–B.II.a.4 (HS**).** Identical to Games A.I.a.0 to A.I.a.4.

Game B.II.a.5 (dHS**)–B.II.a.10.** Identical to Games A.I.a.6 to A.I.a.11.

In this case, the advantage of the adversary in Game 3 is at most

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

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

The security of the KEM_c ciphertext C_c supports the secrecy of the stage keys in this case.

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

Game B.II.b.1 (Client KEM). Similar to Game A.I.a.1, but with K_c and C_c instead of K_s and C_s.

Game B.II.b.2 (MS**).** The challenger of this game replaces the master secret in σ with a uniformly random value. For any session of $\sigma.pid$ that received the ciphertext C_c sent by σ, the master secret MS is also replaced with a uniformly random value, while maintaining consistency across all the sessions that share the same derived intermediate master secret dIMS.

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 session dES (K_c 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_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 B.II.b.3 (SAHTS**,** CAHTS**,** fk_s**,** SATS**,** fk_c **and** CATS**).** These values are now replaced with uniformly random ones in σ. For any session of $\sigma.pid$ that received the ciphertext C_c sent by σ, the master secret MS is also replaced with a uniformly random value, while maintaining consistency across all the sessions that share the same master secret MS. In particular, 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 B.II.b.4 (MAC Forgery). The challenger of this game, running the tested server session, rejects the `ClientFinished` message in case there is no partner session at stage 5. The fact that there is no partner session at stage 5 implies that no honest session computed a MAC tag on the transcript of the tested client 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 3 is in this case at most

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

Case B.III.a: Tested Client with Explicitly Authenticated Server

The main 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 (as in Game B.II.a.1) since the reduction algorithm 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 insight in this case is now to show that under the assumption that the MAC used to compute the `ServerFinished` message is secure, the adversary cannot make the client maliciously accept stage 4 without having first corrupted the server long-term key. That is to say, no client session maliciously accepts stage 4 under the assumption that the MAC is secure. Therefore, if a client session accepts stage 4, then it is guaranteed that 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 4 all retroactively explicitly authenticated, and the keys computed at these stages are indistinguishable from random according to the analysis in case B.I.

Games B.III.a.0–B.III.a.10. Game B.III.a.0 is identical to Game B.II.a.0 and Games B.III.a.1 to B.III.a.9 are defined similarly to Games B.II.a.1 to B.II.a.9, except that the challenger aborts the interaction with the adversary and returns 1 (i.e., the adversary wins) if the tested client session accepts stage 4 without a partner session. Game B.III.a.10 is identical to Game B.II.a.10. Note that in the last game, the client session never accepts stage 4 without a partner session.

In this case, the advantage of the adversary in Game 3 is at most

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

Case B.III.b: Tested Server with Explicitly Authenticated Client

The same reasoning as for client sessions applies here, except that the client explicitly authenticates herself at stage 5. The advantage of the adversary in Game 3 is in this case at most

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

Overall Advantage

The advantage of the adversary in Game 3 is the maximum of its advantage in each of the cases, which is at most

$$n_{id}\begin{pmatrix} \varepsilon^{\text{IND-CCA}}_{\text{KEM}_c} + \varepsilon^{\text{IND-CCA}}_{\text{KEM}_s} \\ + 3\varepsilon^{\text{PRF}}_{\text{HKDF.Extract}} + \varepsilon^{\text{dual-PRF}}_{\text{HKDF.Extract}} \\ + 5\varepsilon^{\text{PRF}}_{\text{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\text{HMAC}} \end{pmatrix} + n_{\sigma}\begin{pmatrix} \varepsilon^{\text{IND-1CCA}}_{\text{KEM}_e} + \varepsilon^{\text{IND-CCA}}_{\text{KEM}_s} \\ + 2\varepsilon^{\text{PRF}}_{\text{HKDF.Extract}} + \varepsilon^{\text{dual-PRF}}_{\text{HKDF.Extract}} \\ + 4\varepsilon^{\text{PRF}}_{\text{HKDF.Expand}} + \varepsilon^{\text{EUF-CMA}}_{\text{HMAC}} \end{pmatrix}.$$

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

$$2^{-257} n_\sigma^2 + \varepsilon_H^{\text{Coll}} + 5n_\sigma \left(\begin{array}{l} n_{id} \left(\begin{array}{l} \varepsilon_{\text{KEM}_c}^{\text{IND-CCA}} + \varepsilon_{\text{KEM}_s}^{\text{IND-CCA}} \\ + 3\varepsilon_{\text{HKDF.Extract}}^{\text{PRF}} + \varepsilon_{\text{HKDF.Extract}}^{\text{dual-PRF}} \\ + 5\varepsilon_{\text{HKDF.Expand}}^{\text{PRF}} + \varepsilon_{\text{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \\ + n_\sigma \left(\begin{array}{l} \varepsilon_{\text{KEM}_e}^{\text{IND-1CCA}} + \varepsilon_{\text{KEM}_s}^{\text{IND-CCA}} \\ + 2\varepsilon_{\text{HKDF.Extract}}^{\text{PRF}} + \varepsilon_{\text{HKDF.Extract}}^{\text{dual-PRF}} \\ + 4\varepsilon_{\text{HKDF.Expand}}^{\text{PRF}} + \varepsilon_{\text{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \end{array} \right).$$

□

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

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

S *is* $\left(T_{\mathsf{S}}^{\mathsf{A}}, q_{\mathsf{S}}^{\mathsf{A}}, \varepsilon_{\mathsf{S}}^{\mathsf{A}}\right)$*-*$\mathsf{A}$*-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*

$$\begin{aligned} \mathscr{O} \in \{&\text{NewSession}, \text{Send}, \text{Reveal}, \text{NextPeriod}, \text{EndCurrentPeriod}, \\ &\text{Corrupt}, \text{SemiStaticCorrupt}, \text{Test}\}. \end{aligned}$$

There exists a real constant $\kappa \leq 1$ *such that if* $T_{\mathscr{A}} + q_{\text{Send}} + q_{\text{Test}} \leq \kappa \min_{(\mathsf{S},\mathsf{A})} \left(T_{\mathsf{S}}^{\mathsf{A}}\right)$ *and if* $n_\sigma := q_{\text{NewSession}} \leq \min_{(\mathsf{S},\mathsf{A})} \left(q_{\mathsf{S}}^{\mathsf{A}}\right)$*, 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 match security game in the case of matching time periods is at most*

$$2^{-257} n_\sigma^2 + \varepsilon_H^{\text{Coll}} + 7n_\sigma \left(\begin{array}{l} n_{id} \left(\begin{array}{l} \varepsilon_{\text{KEM}_c}^{\text{IND-CCA}} + \varepsilon_{\text{KEM}_s}^{\text{IND-CCA}} \\ + 3\varepsilon_{\text{HKDF.Extract}}^{\text{PRF}} + \varepsilon_{\text{HKDF.Extract}}^{\text{dual-PRF}} \\ + 7\varepsilon_{\text{HKDF.Expand}}^{\text{PRF}} + \varepsilon_{\text{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \\ + n_\sigma \left(\begin{array}{l} \varepsilon_{\text{KEM}_e}^{\text{IND-1CCA}} + \varepsilon_{\text{KEM}_s}^{\text{IND-CCA}} \\ + 2\varepsilon_{\text{HKDF.Extract}}^{\text{PRF}} + \varepsilon_{\text{HKDF.Extract}}^{\text{dual-PRF}} \\ + 6\varepsilon_{\text{HKDF.Expand}}^{\text{PRF}} + \varepsilon_{\text{HMAC}}^{\text{EUF-CMA}} \end{array} \right) \end{array} \right),$$

with $n_{id} := |ID|$ *and* $\varepsilon_H^{\text{Coll}}$ *the probability that an algorithm given in the proof finds a collision for* H *by running* $\mathscr{A}$ *as subroutine.*

Proof. The proof is similar to the proof of Theorem 5.2, except that there are seven stages, i.e., two more stage keys (SHTS and CHTS) derived, hence the factor $7n_\sigma$ from the reduction to a single Test query and the extra $2\varepsilon_{\text{HKDF.Expand}}^{\text{PRF}}$ term in the bound. □

5.3 Discussion

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" [31]) 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 [10, 11], 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 [4, 14] is satisfied once the other party is explicitly authenticated.

Comparison with KEMTLS. The assumption that the client knows the server public key from the beginning of the protocol is precisely what allows to reduce the handshake by a full round-trip and have the server send application data from its first message flow, compared to the KEMTLS protocol. It also implies that the server certificate need not be verified 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). 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.

Acknowledgments

This work was supported by the Eurostars ZERO-TOUCH Project (E113920).

References

1. 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.
2. 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.
3. 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.
4. 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.
5. C. Boyd and K. Gellert. A modern view on forward security. Cryptology ePrint Archive, Report 2019/1362, 2019. `https://eprint.iacr.org/2019/1362`.
6. C. Brzuska. *On the Foundations of Key Exchange*. PhD thesis, Technische Universität, Darmstadt, 2013.
7. 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.
8. 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.
9. R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 143–161. Springer, Heidelberg, Aug. 2002. `http://eprint.iacr.org/2002/120/`.
10. 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`.
11. 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`.
12. 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.
13. B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology*, 2021. To appear. Available as Cryptology ePrint Archive, Report 2020/1044. `https://eprint.iacr.org/2020/1044`.
14. 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.
15. Smartm2m; guidelines for security, privacy and interoperability in iot system definition; a concrete approach. Technical Report ETSI SR 003 680, ETSI, 2020.
16. M. Fagan, K. Megas, K. Scarfone, and M. Smith. Foundational cybersecurity activities for iot device manufacturers. Technical Report NISTIR 8259, NIST, 2020.
17. M. Fagan, K. Megas, K. Scarfone, and M. Smith. Iot device cybersecurity capability core baseline. Technical Report NISTIR 8259A, NIST, 2020.
18. 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.

19. 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.
20. F. Günther. *Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols*. PhD thesis, Technische Universität, Darmstadt, 2018.
21. 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.
22. 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.
23. 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.
24. H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. Cryptology ePrint Archive, Report 2015/978, 2015. `http://eprint.iacr.org/2015/978`.
25. K. Kwiatkowski and L. Valenta. The tls post-quantum experiment. https://blog.cloudflare.com/the-tls-post-quantum-experiment/, 2019.
26. 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.
27. A. Langley. Cecpq2. https://www.imperialviolet.org/2018/12/12/cecpq2.html, 2018.
28. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), Aug. 2018.
29. 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 20*, pages 1461–1480. ACM Press, Nov. 2020.
30. K. Sjöberg, P. Andres, T. Buburuzan, and A. Brakemeier. C-ITS deployment in europe - current status and outlook. *CoRR*, abs/1609.03876, 2016.
31. 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.
32. 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.