Post-quantum TLS without handshake signatures

Full version, May 7, 2020

Peter Schwabe
Radboud University
peter@cryptojedi.org

Douglas Stebila
University of Waterloo
dstebila@uwaterloo.ca

Thom Wiggers
Radboud University
thom@thomwiggers.nl

ABSTRACT

We present KEMTLS, an alternative to the TLS 1.3 handshake that uses key-encapsulation mechanisms (KEMs) instead of signatures for server authentication. Among existing post-quantum candidates, signature schemes generally have larger public key/signature sizes compared to the public key/ciphertext sizes of KEMs: by using an IND-CCA-secure KEM for server authentication in post-quantum TLS, we obtain multiple benefits. A size-optimized post-quantum instantiation of KEMTLS requires less than half the bandwidth of a size-optimized post-quantum instantiation of TLS 1.3. In a speed-optimized instantiation, KEMTLS reduces the amount of server CPU cycles by almost 90% compared to TLS 1.3, while at the same time reducing communication size, reducing the time until the client can start sending encrypted application data, and eliminating code for signatures from the server's trusted code base.

KEYWORDS

Post-quantum cryptography, key-encapsulation mechanisms, Transport Layer Security, NIST PQC

1 INTRODUCTION

The Transport Layer Security (TLS) protocol is possibly one of the most-used secure channel protocols. It provides not only a secure way to transfer web pages [76], but is also to secure communications to mail servers [43, 68] or to set up VPN connections [70]. The most recent iteration is TLS 1.3, standardized in August 2018 [77]. The TLS 1.3 handshake uses ephemeral (elliptic curve) Diffie–Hellman (DH) key exchange to establish forward-secret session keys. Authentication of both server and (optionally) client is provided by either RSA or elliptic-curve signatures. Public keys for the signatures are embedded in certificates and transmitted during the handshake. Figure 1 gives a high-level overview of the TLS 1.3 protocol, focusing on the signed-Diffie–Hellman aspect of the handshake.

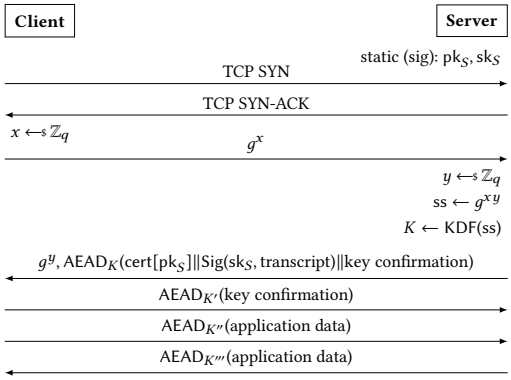

Figure 1: High-level overview of TLS 1.3, using signatures for server authentication. Primes (′) on keys denote additional keys derived from the same shared secret using appropriate labels.

Preparing for post-quantum TLS. There have been many experiments and research in the past five years on moving the TLS ecosystem to post-quantum cryptography. Most of the work has focused on adding post-quantum key exchange to TLS, usually in the context of so-called "hybrid" key exchange that uses both a post-quantum algorithm and a traditional (usually elliptic curve) algorithm, beginning with an experimental demonstration in 2015 of ring-LWE-based key exchange in TLS 1.2 [19].

Public experiments by industry started in 2016 with the CECPQ1 experiment by Google [63], combining X25519 ECDH [9] with NewHope lattice-based key exchange [2] in the TLS 1.2 handshake. A CECPQ2 followup experiment with TLS 1.3 was announced in late 2018 [62, 64] and is currently being run by Google using a combination of X25519 and the lattice-based scheme NTRU-HRSS [45, 46], and by Cloudflare using X25519/NTRU-HRSS and X25519 together with the supersingular-isogeny scheme SIKE [50]. First results from this experiment are presented in [61]. In late 2019, Amazon announced that the AWS Key Management Service (AWS KMS) now supports two ECDH-post-quantum hybrid modes; one also using SIKE, the other one using the code-based scheme BIKE [3].

Additionally, the Open Quantum Safe (OQS) initiative [86] provides prototype integrations of post-quantum and hybrid key exchange in TLS 1.2 and TLS 1.3 via modifications to the OpenSSL library [69]. First results in terms of feasibility of migration and performance using OQS were presented in [29]; more detailed benchmarks are presented in [71].

Draft specifications for hybrid key exchange in TLS 1.3 have already started to appear [54, 87, 89].

Most of the above efforts only target what is often called "transitional security": they focus on quantum-resistant confidentiality using post-quantum key exchange, but not quantum-resistant authentication. The OQS OpenSSL prototypes do support post-quantum authentication in TLS 1.3, and there has been a small amount of research on the efficiency of this approach [82]. While post-quantum algorithms generally have larger public keys, ciphertexts, and signatures compared to pre-quantum elliptic curve schemes, the gap is bigger for post-quantum signatures than post-quantum key encapsulation mechanisms (KEMs).

Authenticated key exchange without signatures. There is a long history of protocols for authenticated key exchange without signatures. *Key transport* uses public key encryption: authentication is demonstrated by successfully decrypting a challenge value. Examples of key transport include the SKEME protocol by Krawczyk [55] and RSA key transport ciphersuites in all versions of SSL and TLS up to TLS version 1.2 (but this did not provide forward secrecy).

Bellare, Canetti, and Rogaway [6] gave a protocol that obtained authentication from Diffie–Hellman key exchange: DH keys are used as *long-term* credentials for authentication, and the resulting shared secret is mixed into the session key calculation to derive a key that is *implicitly authenticated*, meaning that no one but the intended parties could compute it; some protocols go on to obtain *explicit* authentication via some form of key confirmation. Many DH-based AKE protocols have been developed in the literature and some are currently used in a few real-world protocols such as Signal [74], the Noise framework [73], and WireGuard [31].

There are a few constructions that use generic KEMs for AKE, rather than static DH [20, 37]. A slightly modified version of the [37] KEM AKE has recently been used to upgrade the WireGuard handshake to post-quantum security [47]. One might think that the same approach can be used for KEM-based TLS, but there are two major differences between the WireGuard handshake and a TLS handshake. First, the WireGuard handshake is mutually authenticated, while the TLS handshake typically features server-only authentication. Second, and more importantly, the WireGuard handshake assumes that long-term keys are known to the communicating parties in advance, while the distribution of the server's long-term certified key is part of the handshake in TLS, leading to different constraints on the order of messages and number of round trips.

The OPTLS proposal by Krawczyk and Wee [59] aims at a signature-free alternative for the common TLS handshake, with authentication via long-term DH keys. OPTLS was at the heart of early drafts of TLS 1.3, but was dropped in favour of signed-Diffie–Hellman. As pointed out in [60], OPTLS makes use of DH as a non-interactive key exchange (NIKE). First the client sends their ephemeral DH public key, which the server combines with its own long-term secret key to obtain a shared key; the server's reply thus implicitly authenticates the server to the client. Note however that the client speaks first, without knowing the server's public key: a straight-forward adaptation of OPTLS to a post-quantum setting would thus require a post-quantum NIKE. Unfortunately, the only somewhat efficient construction for a post-quantum NIKE is CSIDH [26], which is rather slow and whose concrete security is the subject of intense debate [10, 11, 13, 17, 72]. The obvious workaround when using only KEMs is to increase the number of round trips, but this comes at a steep performance cost.

Our contributions. Our goal is to achieve a TLS handshake that provides full post-quantum security—including confidentiality and authentication—optimizing for number of round trips, communication bandwidth, and computational costs. Our main technique is to rely on KEMs for authentication, rather than signatures.

We present an alternative TLS handshake, which we call KEMTLS, that uses an IND-CCA-secure key-encapsulation mechanism as the primary asymmetric building block, for both forward-secure ephemeral key exchange and authentication. (We unavoidably still rely on signatures by certificate authorities to authenticate long-term KEM keys.) A high level overview of KEMTLS is given in Fig. 2, and the detailed protocol appears in Fig. 5. We focus on the most common use case for web browsing, namely key agreement with server-only authentication, but our techniques can be extended to client authentication as shown in Appendix D.

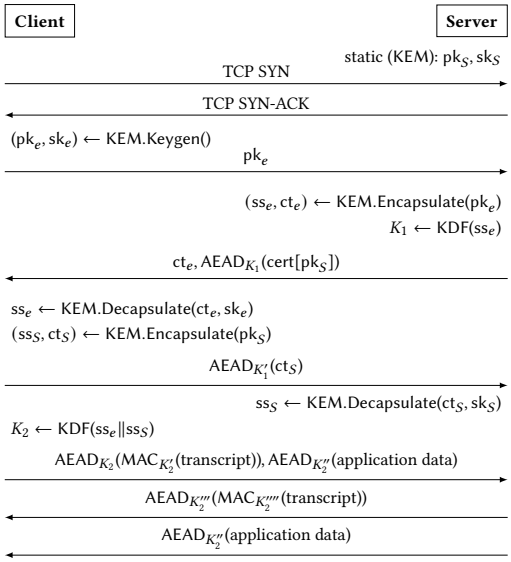

Figure 2: High-level overview of KEMTLS, using KEMs for server authentication.

With KEMTLS, we are able to retain the same number of round trips until the client can start sending encrypted application data as in TLS 1.3 while reducing the communication bandwidth. Although our approach can be applied with any secure KEM, we consider four example scenarios in the paper: (1) optimizing communication size assuming one intermediate certificate is included in transmission, (2) optimizing communication size assuming intermediate certificates can be cached and thus are excluded from transmission, (3) handshakes relying on the module learning with errors (MLWE) / module short-integer-solutions (MSIS) assumptions, and (4) handshakes relying on the NTRU assumption. In all 4 scenarios, KEMTLS is able to reduce communication sizes compared to server authentication using post-quantum signatures.

For example, considering all level-1 schemes in Round 2 of the NIST PQC project, the minimum size of public key cryptography objects transmitted in a fully post-quantum signed-KEM TLS 1.3 handshake that includes transmission of an intermediate certificate would be 3035 bytes (using SIKE for key exchange, Falcon for server authentication, a variant of XMSS for the intermediate CA, and GeMSS for the root CA), whereas with KEMTLS we can reduce that by 39% to 1853 bytes (using SIKE for key exchange and server authentication, a variant of XMSS for the intermediate CA, and GeMSS for the root CA); compare with 840 bytes for RSA-signed elliptic-curve DH in TLS 1.3. Fig. 3 shows the impact of the KEMTLS protocol design on communication sizes for all the scenarios we consider; details appear in Table 1.

To assess computational costs, we implemented KEMTLS by modifying the Rustls library [15], using optimized C/assembly implementations of the relevant post-quantum schemes. We measured performance of this implementation in a range of network scenarios following the methodology of [71], varying latency and bandwidth. We found that KEMTLS results in better client and server performance for scenarios involving the MLWE/MSIS and NTRU assumptions. Our first two scenarios aim to absolutely minimize

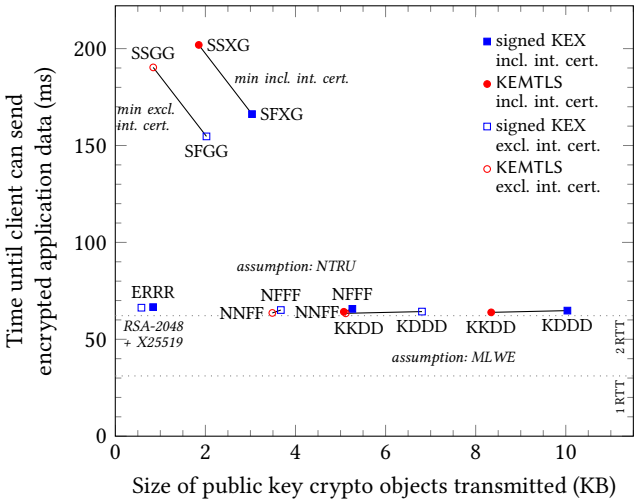

Figure 3: Handshake size versus handshake establishment time, for signed KEX and KEMTLS ciphersuites, including and excluding transmission/processing of one intermediate certificate. Latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Label syntax: ABCD: A = ephemeral key exchange, B = leaf certificate, C = intermediate certificate, D = root certificate. Label values: Dilithium, ECDH X25519, Falcon, GeMSS, Kyber, NTRU, RSA-2048, SIKE, $\text{XMSS}^{\text{MT}}_{\text{s}}$; all level-1 schemes.

communication bandwidth by replacing a fast signature scheme (Falcon) with a smaller but slower KEM (SIKE), which, admittedly, substantially slows down connection establishment, but may still be relevant when communication bandwidth is of utmost concern. See Fig. 3 for an overview and Section 6 for details.

We show that our KEMTLS approach indeed results in a secure protocol, adapting the reductionist security analysis of Dowling, Fischlin, Günther, and Stebila [32, 33] for signed-DH in TLS 1.3. The proof is in the standard model, but like previous proofs of security for TLS [32, 48, 58] we use a nonstandard interactive assumption. In the proofs of TLS 1.3, this interactive assumption is the PRF-ODH assumption first introduced in [48]; the equivalent in our KEM-based handshake is what we call PRF-IND-CCA. We show that PRF-IND-CCA of a KEM follows from OW-CCA, but this reduction uses the random-oracle model (ROM).

Software and data. For the experiments in this paper, we used and modified open-source cryptographic software and TLS libraries. In addition, we wrote new software to facilitate our experiments and to create certificates. All software and data is available at https://thomwiggers.nl/publication/kem-tls/ and https://cryptojedi.org/crypto/#kemtls. All software we modified is under permissive open-source licenses; we place our code into the public domain (CC0).

Discussion. There are a few subtle differences in the properties offered by KEMTLS compared to TLS 1.3. TLS 1.3 allows the server to send encrypted and authentication application data in its first response message, whereas KEMTLS does not. However, in most uses of TLS 1.3, including web browsing, this feature is not used, and the first application data is sent by the client in the second client-to-server TLS message flow, which KEMTLS preserves.

KEMTLS provides *implicit* server-to-client authentication at the time the client sends its first application data; explicit server-to-client authentication comes one round trip later when a key confirmation message is received in the server's response. We still retain confidentiality: no one other than the intended server will be able to read data sent by the client. One consequence is that the server's choice of symmetric ciphersuite is not authenticated by the time client sends its first application data; the client cannot be tricked into using a ciphersuite that it itself does not trust, but an adversary might be able to trick the parties into using one that the server would have rejected. We discuss this subtlety more in Section 6.

Comparison with OPTLS. Our proposal for a signature-free handshake protocol in TLS shares a lot of similarities with the OPTLS protocol [59]. OPTLS was at the heart of early designs for TLS 1.3, but was dropped in favour of signed-DH for the final standard. Starting in 2018, there has been an attempt to revive OPTLS in TLS 1.3 [78, 79], but so far we do not see that these drafts have gained much traction. (The only implementation of OPTLS that we are aware of is described in the Master's thesis by Kuhnen [60].)

If a signature-free approach for the TLS handshake hasn't been very successful in the past, why revisit it now? We see two reasons why OPTLS never gained much traction and both change with the eventual move to post-quantum cryptography in TLS.

To tap the full potential of OPTLS, servers would need to obtain certificates containing DH public keys instead of signature keys; while this is in theory not a problem, it requires certificate authorities to adapt their software and needs other changes to the public-key infrastructure, which would have been obstacle to TLS 1.3's goals of widespread deployment and fast adoption. However, the move to post-quantum authentication will require rolling out a new generation of certificates regardless of whether signatures or KEMs are used for authentication.

Moreover, when using pre-quantum primitives based on elliptic curves, the advantages of OPTLS compared to the traditional TLS 1.3 handshake are limited. The performance differences between ECDH operations and ECDSA or EdDSA signing and verification are not very large, and sizes of signatures and signature public keys are small. A TLS implementation with secure and optimized elliptic-curve arithmetic implemented for ECDH already has most critical code needed to implement ECDSA or EdDSA signatures.

For current post-quantum KEMs and signature schemes, this picture changes. It is possible to choose KEMs that offer considerably smaller sizes and much better speed than any of the signature schemes. Also, post-quantum signatures and KEMs no longer share large parts of the code base; even though lattice assumptions can be used to construct both KEMs and signatures, such schemes need different parameters and thus different optimized routines.

Thus, in the post-quantum setting, the signature-free approach to the TLS handshake offers major advantages. KEMTLS simultaneously reduces the amount of data transmitted during a handshake, reduces the amount of CPU cycles spent on asymmetric crypto, reduces the total handshake time until the client can send application data, and reduces the trusted code base.

2 PRELIMINARIES

In this section we review key encapsulation mechanisms (KEMs) and their security properties, including formulating the notion of PRF-IND-CCA security we employ in one of our proofs.

Notation. Let $\mathbb{N}$ denote the set of natural numbers. For a set X, the notation $x \leftarrow\!\!\$\ X$ denotes sampling an element uniformly at random from the set X and storing it in x. If $\mathcal{A}$ is a deterministic algorithm, then $y \leftarrow \mathcal{A}(x)$ denotes running $\mathcal{A}$ with input x and storing the output in y. If $\mathcal{A}$ is a probabilistic algorithm, then $y \leftarrow\!\!\$\ \mathcal{A}(x)$ denotes running $\mathcal{A}$ with input x and uniformly random coins, and storing the output in y. The notation $[\![x = y]\!]$ resolves to 1 if $x = y$, and 0 otherwise. The TLS protocol has named messages, such as `ClientHello`, which we abbreviate like `CH`, as in Fig. 5.

Symmetric primitives. We rely on standard definitions of symmetric primitives such as hash function with collision resistance, pseudorandom functions, and message authentication codes with existential unforgeability under chosen message attacks, the definitions of which appear in the Appendix A. We do note here the syntax of HKDF [57], which is comprised of two components. HKDF.Expand is a randomness extractor with two inputs: a *salt* and some *input keying material*; in the TLS 1.3 key schedule, the salt argument is used for the current secret state, and the input keying material argument is used for new secret shared secrets being incorporated. HKDF.Expand is a variable-length pseudorandom function with (in this context) four inputs: a secret key, a label, a context string consisting of a hash of a transcript of messages, and the desired output length (which we omit in our presentation).

2.1 KEMs

Definition 2.1 (Key Encapsulation Mechanism (KEM)). A *key encapsulation mechanism* KEM is an asymmetric cryptographic primitive that allows two parties A and B to establish a shared secret key ss in a key space $\mathcal{K}$ to be used for further communication. It consists of the following operations:

- **Key generation:** KEM.Keygen() probabilistically generates a public and private keypair (pk, sk);
- **Encapsulation:** KEM.Encapsulate(pk) probabilistically generates a shared secret and ciphertext (encapsulation) (ss, ct) against a given public key;
- **Decapsulation:** KEM.Decapsulate(ct, sk) decapsulates the shared secret ss′ which, in a correct scheme, is equal to ss.

KEM security notions. There are several standard security definitions for KEMs: that the shared secret is hard to compute (one-way (OW)) or indistinguishable from random (IND), given just the public key (chosen plaintext attack (CPA)) or given access to a decapsulation oracle (chosen ciphertext attack (CCA)). The security experiments for these four security properties are shown in Fig. 4.

For the indistinguishability-based notion (IND-*atk*), the advantage of $\mathcal{A}$ in breaking KEM is $\mathsf{Adv}^{\text{IND-}atk}_{\text{KEM},\mathcal{A}} = \left|\Pr\left[G^{\text{IND-}atk}_{\text{KEM},\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2}\right|$. For the one-way notion (OW-CCA), the advantage of $\mathcal{A}$ in breaking KEM is $\mathsf{Adv}^{\text{OW-CCA}}_{\text{KEM},\mathcal{A}} = \Pr\left[G^{\text{OW-CCA}}_{\text{KEM},\mathcal{A}} \Rightarrow 1\right]$.

2.2 PRF-IND-CCA security of KEMs

Our security analysis of KEMTLS depends on a new interactive security assumption for KEMs, called the PRF-IND-CCA assumption, which is an adaptation of the PRF-ODH assumption used in the security analyses of TLS 1.2 [48, 58], TLS 1.3 [32, 33], the Signal protocol [27], and WireGuard [34].

PRF-ODH. The *pseudorandom function oracle Diffie–Hellman* (PRF-ODH) assumption was introduced by Jager, Kohlar, Schäge, and Schwenk [48] for their analysis of signed-Diffie–Hellman ciphersuites in TLS 1.2 (specifically server-to-client authentication), building on the oracle Diffie–Hellman (ODH) assumption of Abdalla, Bellare, and Rogaway [1]. The PRF-ODH assumption simultaneously concerns a group $\mathbb{G}$ and a pseudorandom function PRF, and requires that the real value $\mathsf{PRF}(g^{uv}, \ell^\star)$, for an adversary-chosen $\ell^\star$, be indistinguishable from a random value, given g^u, g^v, and the ability to learn $\mathsf{PRF}(S^u, \ell)$ and $\mathsf{PRF}(T^v, \ell)$ for adaptively chosen values $S \neq g^v$, $T \neq g^u$, ℓ. Krawczyk, Paterson, and Wee [58] showed that a PRF-ODH-like interactive assumption is in fact necessary for proving server-to-client authentication of TLS 1.2, intuitively due to related-key attacks possible within the security model that must be answered correctly in a reduction.

PRF-IND-CCA. The PRF-IND-CCA assumption is an adaptation of the PRF-ODH assumption (specifically, the snPRF-ODH variant) to a generic KEM setting. Specifically, it demands that the adversary distinguish from random the value $\mathsf{PRF}(\mathsf{ss}^\star, \ell)$ for an adversary-chosen label $\ell^\star$, given the public key $\mathsf{pk}^\star$ and ciphertext $\mathsf{ct}^\star$ encapsulating shared secret $\mathsf{ss}^\star$, and also given access to an oracle that provides outputs of PRF applied to the decapsulation of any ciphertext of the adversary's choosing (other than $\mathsf{ct}^\star$).

The precise specification of the PRF-IND-CCA assumption appears in Fig. 4. The advantage $\mathsf{Adv}^{\text{PRF-IND-CCA}}_{\text{PRF,KEM},\mathcal{A}}$ is defined analogously to that of the IND-CCA advantage.

The PRF-IND-CCA assumption is useful in our security analysis of KEMTLS (Theorem 4.1) because it allows us to modularize part of the security argument. As we show in Theorem 2.2, PRF-IND-CCA security follows from the one-way (OW-CCA) security of the KEM, modelling PRF as a random oracle.

It would be possible to "inline" this OW-CCA+ROM argument into the security proof for KEMTLS, but then that full proof becomes a random oracle model proof. By abstracting via the PRF-IND-CCA property, we can keep the security proof of KEMTLS generic and in the standard model, isolating the random oracle component to the PRF-IND-CCA proof. This leaves open two questions: either (1) proving PRF-IND-CCA security in the standard model (which we have no evidence against, but note that BFGJ [21] showed a negative result concerning PRF-ODH, specifically that there is no algebraic black-box reduction from sn PRF-ODH to a certain class of DDH-related problems), or (2) proving PRF-IND-CCA security in the quantum random oracle model.

Theorem 2.2. *Let* PRF *be a pseudorandom function and* KEM *be a key encapsulation mechanism. Treating* PRF *as a random oracle, if* KEM *is an* OW-CCA*-secure KEM, then the* PRF-IND-CCA *assumption holds. More specifically, let $\mathcal{A}$ be an adversary against the* PRF-IND-CCA *security of* PRF, KEM, *and let q be the number of queries made by $\mathcal{A}$ to the random oracle for* PRF. *Then, for the*

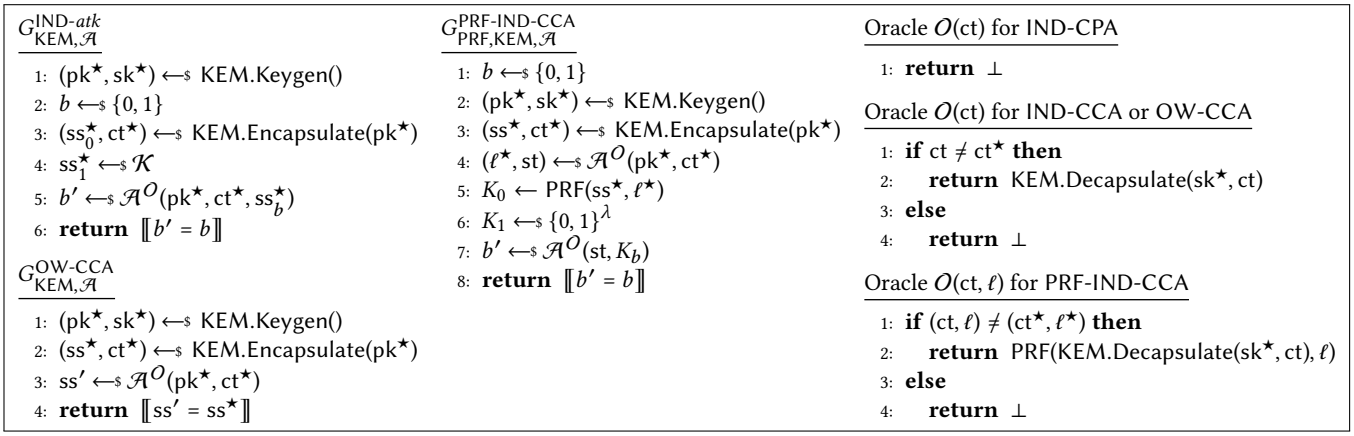

Figure 4: Security experiments for one-way security (OW), indistinguishability (IND), and PRF-indistinguishability (PRF-IND) of KEMs, under chosen plaintext (*atk* = CPA) and chosen ciphertext (*atk* = CCA) attacks.

adversary $\mathcal{B}$ given in Fig. 7, $\text{Adv}^{\text{PRF-IND-CCA}}_{\text{KEM},\mathcal{A}} \leq q \cdot \text{Adv}^{\text{OW-CCA}}_{\text{KEM},\mathcal{B}}$*, and the runtime of $\mathcal{B}$ is the runtime of $\mathcal{A}$ plus a small overhead.*

The idea of the proof is as follows. Since PRF is modelled as a random oracle, $\mathcal{A}$ learns information about the final key only if it queries the shared secret to the random oracle; if it has queried the shared secret to the random oracle, then the solution to the OW-CCA challenge is in the list of random oracle queries. Although we cannot recognize which among several potential ones is the correct solution, returning one at random still solves OW-CCA with non-negligible probability. The full proof is in Appendix B.

2.3 Authenticated key exchange from KEMs

As sketched in the introduction, authenticated key exchange using KEMs for authentication is not new, with several examples of mutually authenticated [18, 20, 37] and unilaterally authenticated [18] protocols. The typical pattern among these, restricted to the case of unilaterally authenticated key exchange, is as follows (c.f. [18, Fig. 2]). The server has a static KEM public key, which the client is assumed to (somehow) have a copy of in advance. In the first flight of the protocol, the client sends a ciphertext encapsulated to this static key, along with the client's own ephemeral KEM public key; the server responds with an encapsulation against the client's ephemeral KEM public key. The session key is the hash of the ephemeral-static and ephemeral-ephemeral shared secrets.

This is a problem for TLS: typically, a client does *not* know the server's static key in advance, but learns it when it is transmitted (inside a certificate) during the TLS handshake. One obvious solution to address this issue is for the client to first request the key from the server and then proceed through the typical protocol flow. However, this increases the number of round trips, and thus comes at a steep performance cost.

The other trivial approach is to simply assume a change in the Internet's key distribution and caching architecture that distributes the servers' static key to the client before the handshake. For example, in embedded applications of TLS, a client may only ever communicate with very few different servers that are known in advance; in that case, the client can just deploy with the server static keys pre-installed. Another option would be to distribute certificates through DNS as described in [51]. Neither is a satisfactory general solution, as the former limits the number of servers a client can contact (since certificates must be pre-installed), and the latter requires changes to the DNS infrastructure and moreover precludes connections to servers identified solely by IP address.

3 THE KEMTLS PROTOCOL

KEMTLS achieves unilaterally authenticated key exchange using solely KEMs for both key establishment and authentication, without requiring extra round trips and without requiring caching or external pre-distribution of server public keys: the client is able to send its first encrypted application data after just as many handshake round trips as in TLS 1.3.

KEMTLS is to a large extent modelled after TLS 1.3. A high-level overview of the handshake is shown in Fig. 2, and a detailed protocol flow is given in Fig. 5. Note that Fig. 5 omits various aspects of the TLS 1.3 protocol that are not relevant to our presentation and cryptographic analysis, such as ciphersuite negotiation, but would still be essential if KEMTLS was used in practice.

There are conceptually three phases to KEMTLS; in each phase, two "stage" keys are established, which figure into the security analysis.

Phase 1: Ephemeral key exchange using KEMs. After establishing the TCP connection, the KEMTLS handshake beings with the client sending an ephemeral KEM public key pk_e in its `ClientHello` message, and the server responds in the `ServerHello` message with an encapsulation ct_e against pk_e. Nonces r_c and r_s are also transmitted for freshness. At this point, the client and server have an unauthenticated shared secret ss_e. KEMTLS follows the TLS 1.3 key schedule, which applies a sequence of HKDF operations to the shared secret ss_e and the transcript to derive (a) the client and server handshake traffic secrets CHTS and SHTS which are used to encrypt subsequent flows in the handshake, and (b) a "derived

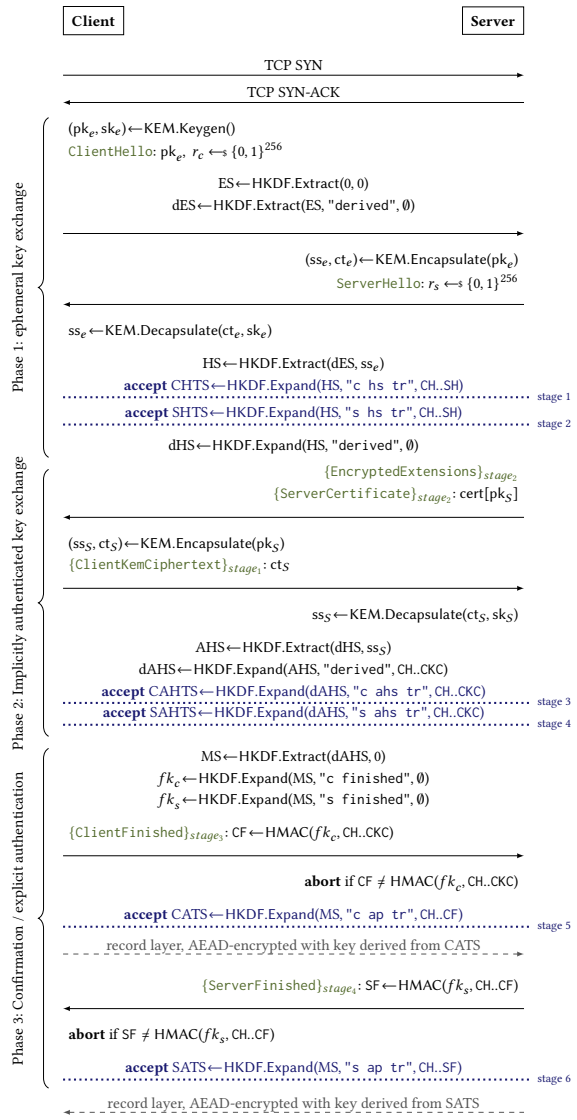

Figure 5: The KEMTLS handshake

handshake secret" dHS which is kept as the current secret state of the key schedule.[1]

Phase 2: Implicitly authenticated key exchange using KEMs. In the same server-to-client flight as `ServerHello`, the server also sends a certificate containing its long-term KEM public key pk_S. The client encapsulates against pk_S and sends the resulting ciphertext in its `ClientKemCiphertext` message. This yields an implicitly authenticated shared secret ss_S. The key schedule's secret state dHS from phase 1 is combined with ss_S using HKDF to give an "authenticated handshake secret" AHS from which are derived (c) the client and server authenticated handshake traffic secrets CAHTS and SAHTS which are used to encrypt subsequent flows in the handshake, and (d) an updated secret state dAHS of the key schedule.

Phase 3: Confirmation / explicit authentication. A master secret MS can now be derived from the key schedule's secret state dAHS. From the master secret, several more keys are derived: (e) "finished keys" fk_c and fk_s which will be used to authenticate the handshake and (f) client and server application transport secrets CATS and SATS from which are derived application encryption keys.[2] The client now sends a confirmation message `ClientFinished` to the server which uses a message authentication code with key fk_c to authenticate the handshake transcript. In the same flight of messages, the client is also able to start sending application data encrypted under keys derived from CATS.

The server responds with its confirmation in the `ServerFinished` message, authenticating the handshake transcript using MAC key fk_s. In the same flight, the server sends application data encrypted under keys derived from SATS. Once the client receives and verifies `ServerFinished`, the server is explicitly authenticated.

4 SECURITY ANALYSIS

As KEMTLS is an adaptation of TLS 1.3, our security analysis follows previous techniques for proving security of TLS 1.3. In particular, we base our approach on the reductionist security approach of Dowling, Fischlin, Günther, and Stebila [32, 33]. Briefly, that approach adapts a traditional Bellare–Rogaway-style [7] authenticated key exchange security model to accommodate multiple *stages* of session keys established in each session, following the multi-stage AKE security model of Fischlin and Günther [35]. The model used for TLS 1.3 in [32, 33] supports a variety of modes and functionality, such as mutual versus unilateral authentication, full handshake and pre-shared key modes, and other options. We are able to simplify the model for our application.

In this section, we give an informal description of the security model, including the adversary interaction (queries) for the model, the specific security properties desired (Match security, which ensures that session identifiers effectively match partnered sessions, and Multi-Stage security, which models confidentiality (indistinguishability of keys established in each stage of a session), and a sketch of the proofs showing that KEMTLS satisfies these properties. The full syntax and specification of the security properties as well as the detailed proofs of security for KEMTLS appear in Appendix C.

Security goal. The main security goal we aim for is that keys established in every stage of KEMTLS should be indistinguishable from a random key, in the face of an adversary who sees and controls all communications, who can learn other stages' keys, who can compromise unrelated secrets (such as long-term keys of parties not involved in the session in question), and who may, after-the-fact, have learned long-term keys of parties involved in the session ("forward secrecy"). This is the same security goal and threat model for TLS 1.3 [32, 77]. In this formulation, entity authentication is not explicitly stated as a security goal: authentication follows implicitly

[1] The key schedule in Fig. 5 starts with a seemingly unnecessary calculation of ES and dES; these values play a role in TLS 1.3 handshakes using pre-shared keys; we retain them to keep the state machine of KEMTLS aligned with TLS 1.3 as much as possible.

[2] TLS 1.3 also derives exporter and resumption master secrets EMS and RMS from the master secret MS. We have omitted these from our presentation of KEMTLS in Fig. 5, but extending KEMTLS's key schedule to include these is straightforward, and security of EMS and RMS follows analogously.

from key indistinguishability. In this section we consider KEMTLS with unilateral server-to-client authentication only, a sketch of KEMTLS with mutual authentication is given in Appendix D.

4.1 Security model

In the following we describe informally the security model, focusing on how it differs from the multi-stage AKE model used by Dowling, Fischlin, Günther, and Stebila [32, 33] to analyze signed-Diffie–Hellman in TLS 1.3. The precise formulation of the model appears in Appendix C.

Model syntax. Each server has a long-term public key and corresponding private key; we assume a public key infrastructure for certifying these public keys, and that the root certificates are pre-distributed, but server certificates are not pre-distributed. Each participant (client or server) can run multiple instances of the protocol, each of which is called a *session*. Note that a session is a participant's local instance of a protocol execution; two parties communicating with each other each have their own sessions. Each session may consist of multiple *stages* (for KEMTLS, there are 6 stages as marked in Fig. 5).

For each session, each participant maintains a collection of session-specific information, including: the identity of the intended communication partner; the *role* of the session owner (either initiator or responder); the *state of execution* (whether it has accepted a key at a certain stage, or is still running, or has rejected); as well as protocol-specific state. For each stage within a session, each participant maintains stage-specific information, including: the *key* established at the stage (if any); a *session identifier* for that stage; and a *contributive identifier* for that stage. Two stages at different parties are considered to partnered if they have the same session identifier. The session identifiers for KEMTLS are the label of the key and the transcript up to that point (see Appendix C.3). For the first stage, the contributive identifier is the `ClientHello` initially, then updated to the `ServerHello` message; for all other stages, the contributive identifier is the session identifier.

The model also records security properties for each stage key:

- Whether the key is considered *authenticated* or not. The model allows for *retroactive* key authentication: the stage-i key may not be considered authenticated at the time it is established in stage i, but may be considered authenticated once stage $j > i$ has completed (e.g., after receiving an additional confirmation message). For KEMTLS, the client considers stage 3–6-keys all authenticated as soon as they are accepted, but the stage-1 and stage-2 keys are considered authenticated only after stage 6 has accepted. We can simplify here compared to the model of DFGS [32, 33] since we only consider server-to-client authentication, not mutual.
- Whether the key is intended for *internal* or *external* use. TLS 1.3 and KEMTLS internally use some of the keys established during the handshake to encrypt later parts of the handshake to improve privacy, whereas other keys are "external outputs" of the handshake to be used for encryption of application data. Internally used keys must be treated more carefully in the security experiment defining key indistinguishability. For KEMTLS, the stage 1–4 keys are for internal use, and the stage 5 and 6 keys are for external use.

Adversary interaction. The adversary is a probabilistic algorithm which triggers parties to execute sessions and controls the communications between all parties, so it can intercept, inject, or drop any message. As a result, the adversary facilitates all interactions, even between honest parties.

The adversary interacts with honest parties via several queries. The first two queries model the typical protocol functionality, which is now under the control of the adversary:

- NewSession: Creates a new session at a party with a specified intended partner and role.
- Send: Delivers a message to a session at a party, which executes the protocol based on its current state, updates its state, and returns any outgoing protocol message.

The next two queries model the adversary's ability to compromise parties' secret information:

- Reveal: Gives the adversary the key established in a particular stage. This key, and the key at the partner session (if it exists), is marked as revealed.
- Corrupt: Gives the adversary a party's long-term secret key. This party is marked as corrupted. To model forward secrecy, all keys established in any stages completed after this query for which the corrupted party is the session owner or partner are marked as revealed.

The final query models the challenge to the adversary of breaking a key established in a stage:

- Test: For a session and stage chosen by the adversary, returns either the real key for that stage, or a uniformly random key, depending on a hidden bit b fixed throughout the experiment.

Some additional conditions apply to the handling of queries. For keys marked as intended for internal use, the execution of the Send query pauses at the moment the key is accepted, giving the adversary the option to either Test that key or continue that execution without testing. This is required since internal keys may be used immediately for, e.g., handshake encryption, and giving the adversary to Test the key after it has already started being used to encrypt data would allow the adversary to trivially win. For keys that are not considered authenticated at the time of the Test query, the query is only permitted if the session has an honest contributive partner, otherwise the adversary could trivially win by active impersonation.

4.2 Security properties

A sequence of works [23, 24, 35] split AKE security into two distinct properties: the traditional session-key indistinguishability property dating back to Bellare and Rogaway [7], and a property they called Match-security, which models the soundness of the session identifier, ensuring that the session identifier π.sid properly matches the partnered π'.sid.

4.2.1 Match *security.* As presented in Appendix C.4, Match-security is defined via a security experiment with the syntax and adversary interaction queries as specified above, followed by checks whether seven technical conditions on session matching are satisfied or not. Roughly speaking, these technical conditions are that stages that are partnered (i.e., which have the same session identifier) (1) derived the same keys at all stages, (2) have opposite initiator/responder

roles, (3) agree on the type of authentication present at each stage, (4) agree on the contributive identifier of each stage, (5) agree on who the session owner and partner are for authenticated stages, (6) have distinct session identifiers from any of their other stages, and (7) are not partnered with an other stages.

For well-chosen session identifiers, proving Match-security typically does not depend on any cryptographic assumptions, and instead follows syntactically. Match-security of KEMTLS is shown in Theorem C.2 in Appendix C.4 (which we omit here). The proof indeed follows syntactically without any cryptographic assumptions, with the sole condition being that nonces in the `ClientHello` and `ServerHello` messages are sufficiently large to make negligible the probability of repeats across sessions by honest parties (which is indeed the case for the 256-bit nonces used in TLS 1.3 and KEMTLS).

4.2.2 Multi-Stage *security.* Secrecy of the key established in each stage is through indistinguishability from random following Bellare–Rogaway [7]. This property, called Multi-Stage security, is defined via an experiment with the syntax and adversary interaction as specified above. The goal of the adversary is to guess the hidden, uniformly random bit b which was used to answer Test queries: was the adversary given real or random keys? As noted above, the experiment imposes constraints on Reveal queries to prevent the adversary from revealing and testing the same key of some stage in a session or its partner, and also constrains the Corrupt query to prevent the adversary from actively impersonating a party in an unauthenticated session then testing that key. We measure the adversary's advantage in guessing b better than just flipping a coin. Details of the experiment appear in Appendix C.5.

The following theorem says that KEMTLS is Multi-Stage-secure with respect to the authentication and internal/external key-use properties as specified above, assuming that the hash function H is collision-resistant, HKDF is a pseudorandom function, HMAC is a secure MAC, KEM is an IND-CPA-secure KEM, and KEM and a function built from HKDF (which we call F, and define in Appendix C.5) together satisfy the PRF-IND-CCA property.

Theorem 4.1. *Let $\mathcal{A}$ be an algorithm, and let n_s be the number of sessions and n_u be the number of parties. Then*

$$\mathsf{Adv}^{\text{Multi-Stage}}_{\text{KEMTLS},\mathcal{A}} \leq 6n_s \left(\begin{array}{l} \epsilon^{\text{COLL}}_{\text{H}} + n_u \left(\begin{array}{l} \epsilon^{\text{PRF-IND-CCA}}_{F} + \epsilon^{\text{PRF-sec}}_{\text{HKDF.Ext}} \\ +2\,\epsilon^{\text{PRF-sec}}_{\text{HKDF.Exp}} + \epsilon^{\text{EUF-CMA}}_{\text{HMAC}} \end{array} \right) \\ +n_s \left(\epsilon^{\text{IND-CPA}}_{\text{KEM}} + 3\,\epsilon^{\text{PRF-sec}}_{\text{HKDF.Ext}} + 4\,\epsilon^{\text{PRF-sec}}_{\text{HKDF.Exp}} \right) \end{array} \right)$$

Above we use the shorthand notation $\epsilon^X_Y = \mathsf{Adv}^X_{Y,\mathcal{B}_i}$ for reductions $\mathcal{B}_i$ that are described in the proof.

The proof of Theorem 4.1 appears in Appendix C.5; here we provide a sketch. The proof proceeds by a sequence of games, and splits into two cases: (A) whether the tested stage has no partner or (B) has a partner.

The Multi-Stage security experiment is formulated to allow the adversary to make multiple Test queries. In the first game hop, we restrict the adversary to make a single Test query by guessing a to-be-tested session using a hybrid argument [41]; this incurs a tightness loss $6n_s$ related to the number of sessions and stages. The second game hope assumes there are no collisions in any hash function calls, which will be useful in later parts of the proof.

The proof then splits into two cases: case A where the (now single) tested session has no honest contributive partner in the first stage; or case B where it does have an honest contributive partner in the first stage.

Case A. Lacking an honest contributive partner in the first stage means the adversary was actively impersonating the peer to the tested session, and there is no partner at any stage of that session. As KEMTLS only provides server-to-client authentication, this means that the tested session in case A is a client session. Moreover, while the stage 3–6-keys are to be considered authenticated immediately when they are accepted, the stage-1 and stage-2 keys are only authenticated after the client receives the `ServerFinished` message and accepts in stage 6. In KEMTLS, the stage 3–6-keys, when accepted, are initially *implicitly authenticated* rather than explicitly authenticated as they would be with signed-Diffie–Hellman.

Within case A, the sequence of game hops is as follows. First, we guess the identity of the server S that the adversary will attempt to impersonate to the client in the tested session.

Then—in the most important step of this case—we replace the dAHS key with a random value. dAHS is derived (via two HKDF calls) from the shared secret ss_S that the client encapsulated against the intended server's long-term static key pk_S. If the KEM is appropriately secure, only the intended server should be able to decapsulate and recover ss_S, and thus ss_S, and the dAHS value derived from it, is an implicitly authenticated key that the adversary should not be able to compute. Formally, this substitution is shown by assuming the PRF-IND-CCA property of the KEM and the two HKDF calls. The reduction PRF-IND-CCA depends on the CCA decapsulation oracle to successfully simulate other sessions involving the server S. Special care must be taken in the reduction to handle the scenario where the adversary might replay the honest client's KEM ciphertext to another session of the same server, which is resolved by how the transcript is mixed into the HKDF calls.

From here, we can replace all keys derived from dAHS with random values in a sequence of games involving PRF-security of HKDF; keys replaced include the stage-3 and stage-4 keys CAHTS and SAHTS, stage-5 and stage-6 keys CATS and SATS, and the finished keys fk_c and fk_s used in the message authentication code. Note that since dAHS is implicitly authenticated at the time it is accepted and indistinguishable from random to the adversary, so are the stage 3–6-keys. The final game in case A makes use of the now-random MAC key to authenticate the transcript of messages received by the client; but since a contribute partner does not exist for stage 1, the honest server S never sent the MAC the client received, and thus we must have a forgery for the MAC scheme, breaking the existential unforgeability of HMAC.

Case B. Here we assume that there *does* exist an honest contributive partner to at least the first stage of the session, i.e., that the adversary did not interfere with the ephemeral key exchange in the `ClientHello` and `ServerHello` messages. Hence, the ephemeral shared secret is unknown to the adversary, assuming an IND-CPA-secure KEM. All keys derived from this are thus also indistinguishable from random, and the remainder of case B is a sequence of game hops which, one-by-one, replace derived secrets with random values, under the PRF-security of HKDF (or, in the derivation of $\text{AHS} \leftarrow \text{HKDF.Extract}(\text{dHS}, \mathsf{ss}_S)$, the "dual PRF security" [5]

of HKDF, since the chaining value is in the "salt" argument of HKDF.Extract, rather than the "input keying material" argument). All stage keys are thus shown indistinguishable from random, yielding the required property for the tested stage in case B.

4.3 Discussion of security properties

We now provide a brief interpretation of the security properties and results from the earlier parts of this section.

Forward secrecy. The security model we use incorporates forward secrecy: a key is a valid target for testing by the adversary as long as it was established prior to the relevant party's long-term secret key being compromised. Intuitively, KEMTLS obtains forward secrecy from its ephemeral key exchange in the `ClientHello` and `ServerHello` messages, and this forward secrecy is retained when the ephemeral shared ss_e (or more specifically, the dHS value derived from ss_e) and the non-forward-secret shared secret ss_S are combined into AHS $\leftarrow$ HKDF.Extract(dHS, ss_S).

The proof only requires IND-CPA security of the ephemeral key exchange to achieve forward secrecy. One might be tempted to use a KEM that only achieves "passive" IND-CPA security for this step, rather than a KEM designed for "active" IND-CCA security, since constructing an IND-CCA secure KEM typically requires adding computationally costly countermeasures like re-encryption using the Fujisaki-Okamoto (FO) transform [38]. However, we still recommend the use of IND-CCA variants even in the ephemeral key exchange. Many IND-CPA-secure post-quantum candidates can be broken with a relatively small number of key reuses. As a result, we believe that the increased robustness from using IND-CCA variants is generally worth the computational cost.

Authentication characteristics. Compared with signed-Diffie–Hellman in TLS 1.2 and TLS 1.3, KEMTLS provides implicit authentication at the time the client starts transmitted encrypted data, rather than explicit authentication. Implicit authentication is not unusual in the key-exchange literature, and it is used in various modern real-world protocols, e.g., the handshakes of Signal and WireGuard.

As shown above, the stage 1 and 2 keys are only retroactively authenticated. This is no different than TLS 1.3: the stage 1 and 2 keys used for handshake encryption provide confidentiality at the time of use only against passive adversaries, but successful completion of the full handshake retroactively demonstrates authentication of the stage 1 and 2 keys.

Tightness. Theorem 4.1 is *non-tight*, due to hybrid and guessing arguments. While it is certainly desirable to have tight results, the few authenticated-key-exchange protocols that have tight proofs have specialized designs, and previous results on TLS 1.3 [32, 33] are similarly non-tight. One can view a non-tight result such as Theorem 4.1 as providing heuristic justification of the soundness of the protocol design, and one can in principle choose parameters for the cryptographic primitives that yield meaningful advantage bounds based on the non-tight reductions.

5 INSTANTIATION AND IMPLEMENTATION

5.1 Choice of primitives

Benchmarking KEMTLS and TLS 1.3 with all combinations of all parameter sets of all NIST PQC candidates would lead to an excessively large set of suites to benchmark. We decided to select 8 different post-quantum suites (4 using TLS 1.3 with signatures, 4 using KEMTLS with only KEMs) that exemplify the following 4 scenarios:

(1) optimizing communication size assuming one intermediate certificate is included in transmission,
(2) optimizing communication size assuming intermediate certificates can be cached and thus excluded from transmission,
(3) handshakes relying on module learning with errors (MLWE) / module short-integer-solutions (MSIS), and
(4) handshakes relying on the NTRU assumption.

We decided on two scenarios focusing on structured lattices (NTRU, MLWE/MSIS) since these give a a very good overall performance in terms of size and speed [61, 71]. The two lattice-based signature schemes Falcon and Dilithium were identified as most efficient for the use in TLS 1.3 in [82]. We contrast these four scenarios against a pre-quantum TLS 1.3 suite using X25519 [9] key exchange with RSA-2048 [80] signatures.

For all primitives we considered the parameter set at NIST security level 1, i.e., targeting security equivalent to AES-128 [36, Sec. 4.A.5]. All primitives we chose are NIST PQC Round 2 candidates, except for an instantiation of the stateful signature algorithm XMSS at NIST level 1 for signatures generated by CAs. XMSS is already defined in an RFC [44] and is being considered by NIST for a fast track to standardization [28]. The XMSS RFC only describes parameters matching NIST level 5 and higher, but the adaptation to a level-1 parameter set is rather straight-forward. We call the level-1 parameter set of XMSS that we use in our experiments $\text{XMSS}_\text{s}^\text{MT}$; details are given in Appendix E. In our scenarios we do not take XMSS as an option for signatures generated by TLS servers, because we do not trust typical TLS servers to securely manage the state, but certificate authorities might be able to do so safely.

Table 1 shows the scenarios and primitives we consider (and the abbreviations we use in the rest of the text to refer to each combination), as well as the resulting communication sizes.

The post-quantum KEMs we use are:

- SIKEp434-compressed [50] as the KEM with the smallest sum of ciphertext and public key;
- Kyber-512 [81] as an overall very efficient KEM based on Module-LWE; and
- NTRU-HPS-2048509 [91] as an overall very efficient KEM based on the NTRU assumption.

The signature schemes we use are:

- GeMSS-128 [25] as the signature scheme with the smallest signature;
- $\text{XMSS}_\text{s}^\text{MT}$ [44], specified in Appendix E, as the signature scheme with the smallest sum of signature and public key;
- Falcon [75] as an overall very efficient signature scheme based on the NTRU assumption and the stateless scheme with the smallest sum of signature and public key; and

Table 1: Instantiations of TLS 1.3 and KEMTLS handshakes with sizes in bytes of transmitted public-key cryptography objects.

			excluding intermediate certificate					intermediate certificate			
		Abbrv.	KEX (pk+ct)	HS auth (ct/sig)	Leaf crt. subject (pk)	Leaf crt. (signature)	Sum excl. int. cert.	Int. crt. subject (pk)	Int. crt. (signature)	Sum incl. int. crt.	CA (pk)
TLS 1.3 (Signed KEX)	**TLS 1.3**	ERRR	X25519 64	RSA-2048 256	RSA-2048 4	RSA-2048 256	**580**	RSA-2048 4	RSA-2048 256	**840**	RSA-2048 4
	Min. incl. int. cert.	SFXG	SIKE 405	Falcon 690	Falcon 897	$XMSS_s^{MT}$ 979	**2971**	$XMSS_s^{MT}$ 32	GeMSS 32	**3035**	GeMSS 352180
	Min. excl. int. cert.	SFGG	SIKE 405	Falcon 690	Falcon 897	GeMSS 32	**2024**	GeMSS 352180	GeMSS 32	**354236**	GeMSS 352180
	Assumption: MLWE+MSIS	KDDD	Kyber 1536	Dilithium 2044	Dilithium 1184	Dilithium 2044	**6808**	Dilithium 1184	Dilithium 2044	**10036**	Dilithium 1184
	Assumption: NTRU	NFFF	NTRU 1398	Falcon 690	Falcon 897	Falcon 690	**3675**	Falcon 897	Falcon 690	**5262**	Falcon 897
KEMTLS	**Min. incl. int. cert.**	SSXG	SIKE 405	SIKE 209	SIKE 196	$XMSS_s^{MT}$ 979	**1789**	$XMSS_s^{MT}$ 32	GeMSS 32	**1853**	GeMSS 352180
	Min. excl. int. cert.	SSGG	SIKE 405	SIKE 209	SIKE 196	GeMSS 32	**842**	GeMSS 352180	GeMSS 32	**353054**	GeMSS 352180
	Assumption: MLWE+MSIS	KKDD	Kyber 1536	Kyber 736	Kyber 800	Dilithium 2044	**5116**	Dilithium 1184	Dilithium 2044	**8344**	Dilithium 1184
	Assumption: NTRU	NNFF	NTRU 1398	NTRU 699	NTRU 699	Falcon 690	**3486**	Falcon 897	Falcon 690	**5073**	Falcon 897

- Dilithium [65] as an overall very efficient scheme based on Module-LWE and Module-SIS.

Caching of intermediate certificates. In order for a client to authenticate a server it typically uses a chain of certificates starting with a certificate whose signature is generated by a root CA, followed by at least one intermediate certificate, and finally the leaf certificate of the actual server. If clients cache the intermediate certificates, those do not need to be transmitted.

The obvious consequences of such caching are that less data is transmitted and that fewer signatures need to be verified. A less obvious consequence is that such caching has significant impact on the optimal choice of (post-quantum) signature scheme for intermediate CAs. If the signed public keys of intermediate CAs are transmitted only once and then cached, what matters most is the size of the signature. This makes multivariate-quadratic-based schemes like LUOV [12], Rainbow [30], or GeMSS [25] with their small signatures but large public keys optimal for use in intermediate certificates. The same logic applies in any case to root CAs, as their corresponding public keys are assumed to be pre-installed.

We investigate both scenarios: *including* transmission and verification of intermediate certificates (i.e., without caching), and *excluding* transmission and verification of intermediate certificates (i.e., with caching). For the including intermediate certificate scenario, we have a single intermediate certificate in the chain.

5.2 Implementation

To experimentally evaluate KEMTLS, we implemented it by modifying Rustls [15], a modern TLS library written in Rust. Rustls provides a clean implementation of TLS 1.3 that was easier to modify than OpenSSL, and provides comparable performance [16]. It uses the Ring [84] library for cryptography and WebPKI [85] for certificate validation. Both of these are also written in Rust, although Ring links to cryptographic implementations from BoringSSL [40].

We first added support for KEM-based key agreement to Ring by changing its ephemeral key-agreement API, designed for Diffie–Hellman key agreement, to a KEM-style API. We updated Rustls to use this new API. Then, we integrated KEMs from PQClean [52], a project that collects cleaned-up implementations of the NIST PQC candidate schemes. Because PQClean provides a standardized, namespaced API, it is straightforward to link together these implementations. We took SIKE and all signature schemes from the Open Quantum Safe (OQS) library [88], although many of the relevant implementations in `liboqs` came from PQClean initially. Where PQClean and OQS did not provide implementations using architecture-specific optimizations (most importantly, AVX2 vector instructions), we ad-hoc integrated those ourselves.

Specifically, we use the AVX2-accelerated code from PQClean for Kyber and Dilithium. The AVX2-optimized implementation of SIKE comes from OQS. We ad-hoc integrated AVX2-accelerated implementations of GeMSS and Falcon, provided by their submitters, into PQClean, and used OQS's scripts to import those into `liboqs`. For $XMSS_s^{MT}$, we used the reference implementation of XMSS, which uses an optimized C implementation of SHAKE-128 for hashing. The pre-quantum and symmetric algorithms are provided by Ring.

To support TLS 1.3 with post-quantum primitives in Rustls, we simply added the KEMs to the list of supported key-exchange algorithms in Rustls. By hard-coding the key share offered by the client, we can then easily force a certain KEM to be used for key exchange. We added the supported signature algorithms to Rustls, Ring, and WebPKI. In various places we needed to update RSA- and EC-inspired assumptions on the sizes of key shares and certificates. For example, Rustls did not expect certificates to be larger than 64 KB, instead of the 16 MB allowed by the RFC [77, App. B.3.3].

Supporting KEMTLS required changing the state machine, for which we modified the TLS 1.3 implementation in Rustls to suit our new handshake. To authenticate using KEM certificates, we added encapsulation and decapsulation using certificates and the corresponding private keys to WebPKI. Rustls and WebPKI do not

support creating certificates. To set up our post-quantum root, intermediate and leaf certificates, we implemented our own certificate generator. It consists of some small programs that create keypairs and signatures. These are used by a Python script that generates the relevant ASN.1 code, signs it, and outputs X.509 certificates.

6 EVALUATION OF KEMTLS vs. TLS 1.3

In this section we compare KEMTLS to TLS 1.3 with post-quantum signatures and key exchange. We first give a comparison in terms of handshake size and speed and then move to properties beyond performance. Finally, we describe consequences for post-quantum signature design when moving to KEMTLS.

6.1 Handshake sizes

Table 1 shows the size of public key cryptographic objects transmitted in KEMTLS versus TLS 1.3.

In scenarios aiming to minimize communication size, switching from TLS 1.3 to KEMTLS can reduce the total number of bytes transmitted in a handshake by 38% from 3035 (SFXG) to 1853 (SSXG), when including intermediate certificates, or by 58% from 2024 (SFGG) to 842 (SSGG), when excluding intermediate certificates.

In scenarios with much faster lattice-based cryptography, switching from TLS 1.3 to KEMTLS also reduces handshake size. For example, when switching TLS 1.3 with Kyber key exchange and Dilithium authentication (KDDD) to KEMTLS with Kyber ephemeral and authenticated key exchange and Dilithium signatures only in certificates (KKDD), handshake size reduces by 16% from 10036 B to 8344 B when including intermediate certificates, and by 24% from 6808 B to 5116 B when excluding intermediate certificates.

6.2 Speed measurements

Benchmarking methodology. In our experiments we use the example TLS client and server implementations provided by Rustls, modifying the client to allow measuring more than one handshake in a loop. We instrument the handshake to print nanoseconds elapsed until operations of interest for both client and server.

We follow the same methodology as [71] for setting up emulated networks. The measurements are done using the Linux kernel's network namespacing [14] and network emulation (`NetEm`) features [42]. We create network namespaces for the clients and the servers and create virtual network interfaces in those namespaces. We vary the latency and bandwidth of the emulate network. `NetEm` adds a latency to the *outgoing* packets, so to add a latency of x ms, we add $\frac{x}{2}$ ms of latency to the client and server interfaces; following [71], we consider round-trip times (RTT) of 31.1 ms (representing an transcontinental connection) and 195.6 ms (representing a trans-Pacific connection). We also throttle the bandwidth of the virtual interfaces, considering both 1000 Mbps and 10 Mbps connections. We do not vary packet loss rate, fixing it at 0%.

We ran measurements on a server with two Intel Xeon Gold 6230 (Cascade Lake) CPUs, each featuring 20 physical cores, which gives us 80 hyperthreaded cores in total. For the measurements, we run forty clients and servers in parallel, such that each process has its own (hyperthreaded) core. We measured 100000 handshakes for each scheme and set of network parameters.

Handshake times. Table 2 shows handshake times for an a high-speed internet connection, with 31.1 ms RTT and 1000 Mbps bandwidth. Table 3 shows handshake times for a slower connection with an RTT of 195.6 ms and a bandwidth limit of 10 Mbps. For both scenarios we highlight in bold-face the time until the client can starting sending encrypted application data; we additionally report the time until the server sends out the `ServerFinished` message and the time when that message has been processed by the client. For TLS 1.3, the latter time is precisely when the client also starts sending data, while in KEMTLS the client sends application data before having processed `ServerFinished`.

For the size-optimized instantiations of KEMTLS, i.e., SSXG and SSGG, we see a slowdown compared to the corresponding SFXG and SFGG instantiations of TLS 1.3, due to the rather high computational cost of the additional usage of compressed SIKE. For the NTRU and module-lattice instantiations, we see a mild increase in speed, which becomes more notable on slower connections. This effect is only to a very small extent due to faster computations, but rather an effect of smaller amounts of data being transmitted.

The handshake times including transmission of intermediate certificates with GeMSS public keys (i.e., SFGG and SSGG) require more round trips, because our benchmarks use the "standard" TCP initial congestion window (`initcwnd`) value of 10MSS. Eliminating this issue would require a massive increase of `initcwnd` to values around 200MSS. See also the discussion in [83, Sec. VII-C].

CPU cycles for asymmetric crypto. For busy Internet servers performing large numbers of TLS handshakes, as well as battery-powered clients, another interesting performance criterion is the computational effort spent on cryptographic operations. For the fast lattice-based schemes the differences in computational effort are not visible from the handshake timings, because computation needs orders of magnitude less time than network communication. We therefore report time in ms required for asymmetric-crypto computations (signing, verifying, key generation, encapsulation, and decapsulation) in Table 4.

As with handshake times, we see the impact of rather slow SIKE key encapsulation and the resulting increase in computational effort when switching from TLS 1.3 with Falcon for authentication (SFXG and SFGG) to KEMTLS with SIKE for authentication (SSXG and SSGG). However, for instantiations with stronger focus on speed, we see a moderate decrease in computational effort on the client side, e.g., 16% when switching from NFFF to NNFF, excluding verification of intermediate certificates. More importantly, we see a massive decrease in computational effort on the server side: saving more than 75% when switching from KDDD to KKDD and almost 90% when switching from NFFF to NNFF.

6.3 Non-performance properties

The handshakes of TLS 1.3 and KEMTLS differ not only in terms of performance but in several other characteristics.

Who can first send application data. In TLS 1.3, the *server* is able to send the first application data after receiving `ClientHello`, i.e., in parallel with its first handshake message to the client and before having received an application-level request from the client. This feature is used, for example, in SMTPS to send a server banner to the client. But this feature is not used in many other applications

Table 2: Average handshake times (in ms) for TLS 1.3 and KEMTLS with 31.1 ms latency and 1000 Mbps bandwidth

		excl. int. crt. Client app. data	excl. int. crt. Client HS done	excl. int. crt. Server HS done	incl. int. crt. Client app. data	incl. int. crt. Client HS done	incl. int. crt. Server HS done
TLS 1.3	**ERRR**	**66.3**	66.3	35.4	**66.6**	66.6	35.6
	SFXG	**165.9**	165.9	134.1	**166.2**	166.2	134.4
	SFGG	**154.7**	154.7	122.9	**377.5**	377.5	345.8
	KDDD	**64.3**	64.3	33.3	**64.8**	64.8	33.9
	NFFF	**65.1**	65.1	34.2	**65.7**	65.7	34.7
KEMTLS	**SSXG**	**201.4**	268.2	205.0	**201.9**	268.6	205.5
	SSGG	**190.3**	256.5	193.3	**406.2**	471.3	408.5
	KKDD	**63.4**	95.7	33.4	**63.9**	110.6	48.3
	NNFF	**63.6**	95.2	32.9	**64.2**	96.5	34.2

Table 3: Average handshake times (in ms) for TLS 1.3 and KEMTLS with 195.6 ms latency and 10 Mbps bandwidth

		excl. int. crt. Client app. data	excl. int. crt. Client HS done	excl. int. crt. Server HS done	incl. int. crt. Client app. data	incl. int. crt. Client HS done	incl. int. crt. Server HS done
TLS 1.3	**ERRR**	**397.1**	397.1	201.2	**398.2**	398.2	202.3
	SFXG	**482.8**	482.8	286.5	**482.6**	482.6	286.2
	SFGG	**474.2**	474.2	278.1	**10939.6**	10939.6	10390.2
	KDDD	**411.7**	411.7	445.9	**416.5**	416.5	448.3
	NFFF	**398.1**	398.1	227.3	**406.8**	406.8	443.4
KEMTLS	**SSXG**	**506.0**	732.3	340.0	**506.3**	732.6	340.3
	SSGG	**496.8**	723.0	330.8	**10858.8**	11860.2	10331.9
	KKDD	**400.9**	837.9	440.8	**419.8**	865.4	447.5
	NNFF	**396.1**	593.3	200.5	**403.4**	841.1	441.9

of TLS, including the most prominent one, HTTPS. In KEMTLS, it is the *client* that is ready to send application data first. This does incur a small overhead in protocols that require a client to receive, for example, a server banner. However, for most typical application scenarios, including HTTPS, in which the client sends a request before receiving any data from the server, this is not a problem.

Ciphersuite for first client data. In KEMTLS, the symmetric ciphersuite that the client uses to send application data before having received the `ServerFinished` message is not authenticated. This allows an attacker to choose the ciphersuite used by the client, however *only from the list of ciphersuites the client is willing to use*. Picking a ciphersuite from a list that the client advertised can hardly be considered a significant attack, especially against careful implementations with promptly disable obsolete algorithms. However, one could address this by fixing the ciphersuite used by the client before having received `ServerFinished` to one of the ciphersuites that every server MUST support as specified in the RFC. For example, for TLS 1.3, "in absence of an application profile standard specifying otherwise" every client and server must support `TLS_AES_128_GCM_SHA256` [77, Sec. 9.1].

Deniability. Already in the mid '90s, Krawczyk pointed out [55, Sec. 2.3.2] that using signatures for explicit authentication in key-agreement protocols adds an unnecessary and undesirable property: non-repudiation. In TLS 1.3, a client is able to use the handshake signatures to prove that a particular server communicated with the client, which might violate the server's privacy goals. This becomes even more of an issue with client authentication in TLS 1.3: the server is able to prove that a particular client communicated with it. The KEM-authenticated handshake of KEMTLS is more privacy friendly because it features deniable authentication [4].

Table 4: Computation time (in ms) on client and server for asymmetric crypto for TLS 1.3 and KEMTLS.

		excl. int. crt. Client	excl. int. crt. Server	incl. int. crt. Client	incl. int. crt. Server
TLS 1.3	**ERRR**	0.134	0.629	0.150	0.629
	SFXG	40.058	21.676	40.094	21.676
	SFGG	34.104	21.676	34.141	21.676
	KDDD	0.080	0.087	0.111	0.087
	NFFF	0.141	0.254	0.181	0.254
KEMTLS	**SSXG**	61.456	41.712	61.493	41.712
	SSGG	55.503	41.712	55.540	41.712
	KKDD	0.060	0.021	0.091	0.021
	NNFF	0.118	0.027	0.158	0.027

Smaller TCB in core handshake. The core KEMTLS handshake is free of signatures, which reduces the trusted code base. Notably, KEMTLS servers no longer need efficient and secure implementations of signing, a routine that has been the target of various side-channel attacks [8, 22, 39, 49, 90]. With KEMTLS, signatures are only generated in the more confined and secured environment of certificate authorities. The effect is less notable on the client side, because clients still need code to verify signatures in certificates. However, this code does not deal with any secret data and thus does not need side-channel protection. The same argument applies for servers in KEMTLS with client authentication as in Appendix D.

6.4 Requirements for post-quantum signatures

Many post-quantum signature schemes can tweak parameters to make different tradeoffs between signature size, signing speed, public-key size, and verification speed. One common direction to optimize for is signing speed, or more precisely signing *latency* reported as the number of clock cycles for a single signature. The common motivation for this optimization is the use of online signatures in handshake protocols like the one used in TLS 1.3 (and earlier versions) or the SIGMA handshake approach [56] used, for example, in the Internet key-exchange protocol (IKE) [53].

In KEMTLS, signatures are only needed for certificates and thus computed offline. This eliminates the requirement for low-latency signing; what remains important (depending to some extent on certificate-caching strategies) is signature size, public-key size, verification latency, and—at least for certificate authorities—signing *throughput*. However, throughput can easily be achieved for any signature scheme by signing the root of an XMSS or LMS tree and using the leaves of that tree to sign a batch of messages. See [66, Sec. 6] and the XMSS discussion in Appendix E.

7 CONCLUSION AND FUTURE WORK

In this paper we presented KEMTLS, an alternative approach to the TLS handshake using an IND-CCA-secure KEM for both key exchange and authentication, which yields significant advantages in terms of communication size and performance compared to TLS 1.3 with post-quantum signatures.

Our analysis only considered the worst-case scenario for KEMTLS, i.e., the scenario in which a client has no prior knowledge of the server's certificate when establishing a connection. This is currently the common case for HTTPS, but there are multiple other applications of TLS that could benefit even more from switching to KEMTLS, where a servers' public keys *are* known to clients. One example are e-mail clients connecting to the same SMTP server every time and keeping a local copy of the server's public key. Other examples are VPN or embedded applications, where clients also communicate only with a very small set of servers. Investigating how KEMTLS behaves in these scenarios, and how to best optimize the choice of algorithms, is an interesting question for future work.

This paper reports performance numbers only for select NIST PQC Round 2 candidates, and only for their parameter sets at NIST security level 1. These instantiations include some of the most promising candidates as identified by prior work [61, 82], but it will be interesting to expand benchmarks to cover KEMTLS and TLS 1.3 with a larger set of primitives and parameter sets.

Finally, our proof of Theorem 2.2 requires the random-oracle model. We leave it to future work to investigate if a reduction exists also in the quantum-random-oracle model (or the standard model).

ACKNOWLEDGMENTS

The authors gratefully acknowledge insightful discussions with Nick Sullivan. This work has been supported by the European Research Council through Starting Grant No. 805031 (EPOQUE) and the Natural Sciences and Engineering Research Council of Canada through Discovery grant RGPIN-2016-05146 and a Discovery Accelerator Supplement.

REFERENCES

[1] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. 2001. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In *CT-RSA 2001 (LNCS)*, David Naccache (Ed.), Vol. 2020. Springer, Heidelberg, 143–158. https://doi.org/10.1007/3-540-45353-9_12

[2] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. 2016. Post-quantum Key Exchange - A New Hope. In *USENIX Security 2016*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 327–343.

[3] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, and Valentin Vasseur. 2019. *BIKE*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[4] Yonatan Aumann and Michael O. Rabin. 1998. IACR Distinguished Lecture: Authentication, Enhanced Security and Error Correcting Codes (Extended Abstract). In *CRYPTO'98 (LNCS)*, Hugo Krawczyk (Ed.), Vol. 1462. Springer, Heidelberg, 299–303. https://doi.org/10.1007/BFb0055736

[5] Mihir Bellare. 2006. New Proofs for NMAC and HMAC: Security without Collision-Resistance. In *CRYPTO 2006 (LNCS)*, Cynthia Dwork (Ed.), Vol. 4117. Springer, Heidelberg, 602–619. https://doi.org/10.1007/11818175_36

[6] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. 1998. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols (Extended Abstract). In *30th ACM STOC*. ACM Press, 419–428. https://doi.org/10.1145/276698.276854

[7] Mihir Bellare and Phillip Rogaway. 1994. Entity Authentication and Key Distribution. In *CRYPTO'93 (LNCS)*, Douglas R. Stinson (Ed.), Vol. 773. Springer, Heidelberg, 232–249. https://doi.org/10.1007/3-540-48329-2_21

[8] Naomi Benger, Joop van de Pol, Nigel P. Smart, and Yuval Yarom. 2014. "Ooh Aah... Just a Little Bit": A Small Amount of Side Channel Can Go a Long Way. In *CHES 2014 (LNCS)*, Lejla Batina and Matthew Robshaw (Eds.), Vol. 8731. Springer, Heidelberg, 75–92. https://doi.org/10.1007/978-3-662-44709-3_5

[9] Daniel J. Bernstein. 2006. Curve25519: New Diffie-Hellman Speed Records. In *PKC 2006 (LNCS)*, Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin (Eds.), Vol. 3958. Springer, Heidelberg, 207–228. https://doi.org/10.1007/11745853_14

[10] Daniel J. Bernstein. 2019. Re: [pqc-forum] new quantum cryptanalysis of CSIDH. Posting to the NIST pqc-forum mailing list. https://groups.google.com/a/list.nist.gov/forum/#!original/pqc-forum/svm1kDy6c54/0gFOLitbAgAJ.

[11] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. 2019. Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies. In *EUROCRYPT 2019, Part II (LNCS)*, Yuval Ishai and Vincent Rijmen (Eds.), Vol. 11477. Springer, Heidelberg, 409–441. https://doi.org/10.1007/978-3-030-17656-3_15

[12] Ward Beullens, Bart Preneel, Alan Szepieniec, and Frederik Vercauteren. 2019. *LUOV*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[13] Jean-François Biasse, Annamaria Iezzi, and Michael J. Jacobson Jr. 2018. A Note on the Security of CSIDH. In *INDOCRYPT 2018 (LNCS)*, Debrup Chakraborty and Tetsu Iwata (Eds.), Vol. 11356. Springer, Heidelberg, 153–168. https://doi.org/10.1007/978-3-030-05378-9_9

[14] Eric W. Biederman and Nicolas Dichtel. 2013. http://man7.org/linux/man-pages/man8/ip-netns.8.html `man ip netns`.

[15] Joseph Birr-Pixton. [n.d.]. A modern TLS library in Rust. https://github.com/ctz/rustls (accessed 2020-04-23).

[16] Joseph Birr-Pixton. 2019. *TLS performance: rustls versus OpenSSL*. https://jbp.io/2019/07/01/rustls-vs-openssl-performance.html

[17] Xavier Bonnetain and André Schrottenloher. 2020. Quantum Security Analysis of CSIDH. In *Advances in Cryptology – EUROCRYPT 2020 (LNCS)*, Anne Canteaut and Yuval Ishai (Eds.), Vol. 12106. Springer, 493–522. https://eprint.iacr.org/2018/537.

[18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. 2018. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018*. IEEE, 353–367. http://cryptojedi.org/papers/#kyber.

[19] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. 2015. Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 553–570. https://doi.org/10.1109/SP.2015.40

[20] Colin Boyd, Yvonne Cliff, Juan Manuel Gonzalez Nieto, and Kenneth G. Paterson. 2009. One-round key exchange in the standard model. *IJACT* 1 (2009), 181–199. Issue 3.

[21] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. 2017. PRF-ODH: Relations, Instantiations, and Impossibility Results. In *CRYPTO 2017, Part III (LNCS)*, Jonathan Katz and Hovav Shacham (Eds.), Vol. 10403. Springer, Heidelberg, 651–681. https://doi.org/10.1007/978-3-319-63697-9_22

[22] Billy Bob Brumley and Nicola Tuveri. 2011. Remote Timing Attacks Are Still Practical. In *ESORICS 2011 (LNCS)*, Vijay Atluri and Claudia Díaz (Eds.), Vol. 6879. Springer, Heidelberg, 355–371. https://doi.org/10.1007/978-3-642-23822-2_20

[23] Christina Brzuska. 2013. *On the foundations of key exchange*. Ph.D. Dissertation. Technische Universität Darmstadt, Darmstadt, Germany. https://tuprints.ulb.tu-darmstadt.de/3414/.

[24] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. 2011. Composability of Bellare-Rogaway key exchange protocols. In *ACM CCS 2011*, Yan Chen, George Danezis, and Vitaly Shmatikov (Eds.). ACM Press, 51–62. https://doi.org/10.1145/2046707.2046716

[25] A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. 2019. *GeMSS*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[26] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. 2018. CSIDH: An Efficient Post-Quantum Commutative Group Action. In *ASIACRYPT 2018, Part III (LNCS)*, Thomas Peyrin and Steven Galbraith (Eds.), Vol. 11274. Springer, Heidelberg, 395–427. https://doi.org/10.1007/978-3-030-03332-3_15

[27] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2017. A formal security analysis of the Signal messaging protocol. In *Proc. IEEE European Symposium on Security and Privacy (EuroS&P) 2017*. IEEE.

[28] David Cooper, Daniel Apon, Quynh Dang, Michael Davidson, Morris Dworkin, and Carl Miller. 2019. *SP 800-208 (Draft) – Recommendation for Stateful Hash-Based Signature Schemes*. Technical Report. NIST. https://csrc.nist.gov/publications/detail/sp/800-208/draft.

[29] Eric Crockett, Christian Paquin, and Douglas Stebila. 2019. Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. Workshop Record of the Second PQC Standardization

Conference. https://csrc.nist.gov/CSRC/media/Events/Second-PQC-Standardization-Conference/documents/accepted-papers/stebila-prototyping-post-quantum.pdf.

[30] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang. 2019. *Rainbow*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[31] Jason A. Donenfeld. 2017. WireGuard: Next Generation Kernel Network Tunnel. In *NDSS 2017*. The Internet Society.

[32] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2015. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, 1197–1210. https://doi.org/10.1145/2810103.2813653

[33] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2016. A Cryptographic Analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol. Cryptology ePrint Archive, Report 2016/081. http://eprint.iacr.org/2016/081.

[34] Benjamin Dowling and Kenneth G. Paterson. 2018. A Cryptographic Analysis of the WireGuard Protocol. In *ACNS 18 (LNCS)*, Bart Preneel and Frederik Vercauteren (Eds.), Vol. 10892. Springer, Heidelberg, 3–21. https://doi.org/10.1007/978-3-319-93387-0_1

[35] Marc Fischlin and Felix Günther. 2014. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *ACM CCS 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM Press, 1193–1204. https://doi.org/10.1145/2660267.2660308

[36] National Institute for Standards and Technology. 2016. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[37] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. 2012. Strongly Secure Authenticated Key Exchange from Factoring, Codes, and Lattices. In *PKC 2012 (LNCS)*, Marc Fischlin, Johannes Buchmann, and Mark Manulis (Eds.), Vol. 7293. Springer, Heidelberg, 467–484. https://doi.org/10.1007/978-3-642-30057-8_28

[38] Eiichiro Fujisaki and Tatsuaki Okamoto. 1999. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *CRYPTO'99 (LNCS)*, Michael J. Wiener (Ed.), Vol. 1666. Springer, Heidelberg, 537–554. https://doi.org/10.1007/3-540-48405-1_34

[39] Cesar Pereida García, Billy Bob Brumley, and Yuval Yarom. 2016. "Make Sure DSA Signing Exponentiations Really are Constant-Time". In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1639–1650. https://doi.org/10.1145/2976749.2978420

[40] Google. [n.d.]. *BoringSSL*. https://boringssl.googlesource.com/boringssl/

[41] Felix Günther. 2018. *Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols*. Ph.D. Dissertation. Technische Universität Darmstadt, Darmstadt, Germany. https://tuprints.ulb.tu-darmstadt.de/7162

[42] Stephen Hemminger, Fabio Ludovici, and Hagen Paul Pfeiffer. 2011. http://man7.org/linux/man-pages/man8/tc-netem.8.html `man ip netem`.

[43] Paul E. Hoffman. 2002. SMTP Service Extension for Secure SMTP over Transport Layer Security. IETF RFC 3207. https://rfc-editor.org/rfc/rfc3207.txt.

[44] Andreas H ulsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. 2018. XMSS: eXtended Merkle Signature Scheme. IETF RFC 8391. https://rfc-editor.org/rfc/rfc8391.txt.

[45] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. 2017. High-Speed Key Encapsulation from NTRU. In *CHES 2017 (LNCS)*, Wieland Fischer and Naofumi Homma (Eds.), Vol. 10529. Springer, Heidelberg, 232–252. https://doi.org/10.1007/978-3-319-66787-4_12

[46] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. 2017. NTRU-KEM-HRSS17: Algorithm Specification and Supporting Documentation. Submission to the NIST Post-Quantum Cryptography Standardization Project. https://cryptojedi.org/papers/#ntrukemnist.

[47] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Philip R. Zimmermann. 2020. Post-quantum WireGuard. Cryptology ePrint Archive, Report 2020/379. http://eprint.iacr.org/2020/379.

[48] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. 2012. On the Security of TLS-DHE in the Standard Model. In *CRYPTO 2012 (LNCS)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.), Vol. 7417. Springer, Heidelberg, 273–293. https://doi.org/10.1007/978-3-642-32009-5_17

[49] Jan Jancar, Petr Svenda, and Vladimir Sedlacek. 2019. Minerva: Lattice attacks strike again. https://minerva.crocs.fi.muni.cz/ (accessed 2020-04-30).

[50] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, and Geovandro Pereira. 2019. *SIKE*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[51] Simon Josefsson. 2006. Storing Certificates in the Domain Name System (DNS). IETF RFC 4398. https://rfc-editor.org/rfc/rfc4398.txt.

[52] Matthias Kannwischer, Joost Rijneveld, Peter Schwabe, Douglas Stebila, and Thom Wiggers. [n.d.]. *PQClean: clean, portable, tested implementations of post-quantum cryptography*. https://github.com/pqclean/pqclean

[53] Charlie Kaufmann, Paul E. Hoffman, Yoav Nir, and Pasi Eronen. 2014. Internet Key Exchange Protocol Version 2 (IKEv2). IETF RFC 7296. https://rfc-editor.org/rfc/rfc7296.txt.

[54] Franziskus Kiefer and Krzysztof Kwiatkowski. 2018. *Hybrid ECDHE-SIDH Key Exchange for TLS*. Internet-Draft draft-kiefer-tls-ecdhe-sidh-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-kiefer-tls-ecdhe-sidh-00 Work in Progress.

[55] Hugo Krawczyk. 1996. SKEME: A Versatile Secure Key Exchange for Internet. In *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*. IEEE, 114–127. http://www.di-srv.unisa.it/ ads/corso-security/www/CORSO-9900/oracle/skeme.pdf.

[56] Hugo Krawczyk. 2003. SIGMA: The "SIGn-and-MAc" Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In *CRYPTO 2003 (LNCS)*, Dan Boneh (Ed.), Vol. 2729. Springer, Heidelberg, 400–425. https://doi.org/10.1007/978-3-540-45146-4_24

[57] Hugo Krawczyk. 2010. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *CRYPTO 2010 (LNCS)*, Tal Rabin (Ed.), Vol. 6223. Springer, Heidelberg, 631–648. https://doi.org/10.1007/978-3-642-14623-7_34

[58] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. 2013. On the Security of the TLS Protocol: A Systematic Analysis. In *CRYPTO 2013, Part I (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8042. Springer, Heidelberg, 429–448. https://doi.org/10.1007/978-3-642-40041-4_24

[59] Hugo Krawczyk and Hoeteck Wee. 2017. The OPTLS Protocol and TLS 1.3. In *Proc. IEEE European Symposium on Security and Privacy (EuroS&P) 2016*. IEEE. https://eprint.iacr.org/2015/978.pdf.

[60] Wouter Kuhnen. 2018. *OPTLS revisited*. Master's thesis. Radboud University. https://www.ru.nl/publish/pages/769526/thesis-final.pdf.

[61] Krzysztof Kwiatkowski, Nick Sullivan, Adam Langley, Dave Levin, and Alan Mislove. 2019. Measuring TLS key exchange with post-quantum KEM. Workshop Record of the Second PQC Standardization Conference. https://csrc.nist.gov/CSRC/media/Events/Second-PQC-Standardization-Conference/documents/accepted-papers/kwiatkowski-measuring-tls.pdf.

[62] Kris Kwiatkowski and Luke Valenta. 2019. The TLS Post-Quantum Experiment. Post on the Cloudflare blog. https://blog.cloudflare.com/the-tls-post-quantum-experiment/.

[63] Adam Langley. 2016. CECPQ1 results. Blog post. https://www.imperialviolet.org/2016/11/28/cecpq1.html.

[64] Adam Langley. 2018. CECPQ2. Blog post. https://www.imperialviolet.org/2018/12/12/cecpq2.html.

[65] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2019. *CRYSTALS-DILITHIUM*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[66] David McGrew, Panos Kampanakis, Scott Fluhrer, Stefan-Lukas Gazdag, Denis Butin, and Johannes Buchmann. 2016. State Management for Hash-Based Signatures. In *Security Standardisation Research (LNCS)*, Lidong Chen, David McGrew, and Chris Mitchell (Eds.), Vol. 10074. Springer, 244–260. https://eprint.iacr.org/2016/357.pdf.

[67] National Institute of Standards and Technology 2015. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. National Institute of Standards and Technology. Federal Information Processing Standards Publication 202, http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[68] Chris Newman. 1999. Using TLS with IMAP, POP3 and ACAP. IETF RFC 2595. https://rfc-editor.org/rfc/rfc2595.txt.

[69] OpenSSL. [n.d.]. OpenSSL: The Open Source toolkit for SSL/TLS. http://www.openssl.org/ (accessed 2020-04-23).

[70] OpenVPN [n.d.]. OpenVPN Protocol. https://openvpn.net/community-resources/openvpn-protocol/ (accessed 2020-03-30).

[71] Christian Paquin, Douglas Stebila, and Goutam Tamvada. 2020. Benchmarking Post-quantum Cryptography in TLS. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, Jintai Ding and Jean-Pierre Tillich (Eds.). Springer, Heidelberg, 72–91. https://doi.org/10.1007/978-3-030-44223-1_5

[72] Chris Peikert. 2020. He Gives C-Sieves on the CSIDH. In *Advances in Cryptology – EUROCRYPT 2020 (LNCS)*, Anne Canteaut and Yuval Ishai (Eds.), Vol. 12106. Springer, 463–492. https://eprint.iacr.org/2018/537.

[73] Trevor Perrin. [n.d.]. Noise Protocol Framework. http://noiseprotocol.org/ (accessed 2020-05-01).

[74] Trevor Perrin and Moxie Marlinspike. 2016. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[75] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2019. *FALCON*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[76] Eric Rescorla. 2000. HTTP over TLS. IETF RFC 2818. https://rfc-editor.org/rfc/rfc2818.txt.

[77] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. IETF RFC 8446. https://rfc-editor.org/rfc/rfc8446.txt.

[78] Eric Rescorla and Nick Sullivan. 2018. *Semi-Static Diffie-Hellman Key Establishment for TLS 1.3.* Internet-Draft. Internet Engineering Task Force. https://tools.ietf.org/html/draft-rescorla-tls13-semistatic-dh-00.

[79] Eric Rescorla, Nick Sullivan, and Christopher A. Wood. 2020. *Semi-Static Diffie-Hellman Key Establishment for TLS 1.3.* Internet-Draft. Internet Engineering Task Force. https://tools.ietf.org/html/draft-rescorla-tls-semistatic-dh-02.

[80] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21 (1978), 120–126.

[81] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. 2019. *CRYSTALS-KYBER.* Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[82] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. Post-Quantum Authentication in TLS 1.3: A Performance Study. In *NDSS 2020.* The Internet Society.

[83] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. 2020. Post-Quantum Authentication in TLS 1.3: A Performance Study. Cryptology ePrint Archive, Report 2020/071. http://eprint.iacr.org/2020/071, updated version of [82].

[84] Brian Smith. [n.d.]. *Ring.* https://github.com/briansmith/ring

[85] Brian Smith. [n.d.]. *WebPKI.* https://github.com/briansmith/webpki

[86] Douglas Stebila and Michele Mosca. 2016. Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. In *SAC 2016 (LNCS)*, Roberto Avanzi and Howard M. Heys (Eds.), Vol. 10532. Springer, Heidelberg, 14–37. https://doi.org/10.1007/978-3-319-69453-5_2

[87] Douglas Steblia, Scott Fluhrer, and Shay Gueron. 2020. *Hybrid key exchange in TLS 1.3.* Internet-Draft draft-ietf-tls-hybrid-design-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-00 Work in Progress.

[88] the Open Quantum Safe project. [n.d.]. *Open Quantum Safe.* https://openquantumsafe.org

[89] William Whyte, Zhenfei Zhang, Scott Fluhrer, and Oscar Garcia-Morchon. 2017. *Quantum-Safe Hybrid (QSH) Key Exchange for Transport Layer Security (TLS) version 1.3.* Internet-Draft draft-whyte-qsh-tls13-06. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-whyte-qsh-tls13-06 Work in Progress.

[90] Yuval Yarom, Daniel Genkin, and Nadia Heninger. 2017. CacheBleed: a timing attack on OpenSSL constant-time RSA. *Journal of Cryptographic Engineering* 7, 2 (June 2017), 99–112. https://doi.org/10.1007/s13389-017-0152-y

[91] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, William Whyte, John M. Schanck, Andreas Hulsing, Joost Rijneveld, Peter Schwabe, and Oussama Danba. 2019. *NTRUEncrypt.* Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

A CRYPTOGRAPHIC DEFINITIONS

KEMTLS depends on several symmetric cryptographic primitives and standard security definitions thereof.

Definition A.1 (Hash function and collision resistance). A *hash function* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ maps arbitrary-length messages $m \in \{0,1\}^*$ to a hash value $\mathsf{H}(m) \in \{0,1\}^\lambda$ of fixed length $\lambda \in \mathbb{N}$. The *collision resistance* of a hash function H measures the ability of an adversary $\mathcal{A}$ to find two distinct messages that hash to the same output:

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

Definition A.2 (Pseudorandom function). A *pseudorandom function* $\mathsf{PRF} : \mathcal{K} \times \mathcal{L} \to \{0,1\}^\lambda$ maps a key $k \in \mathcal{K}$ and a label $\ell \in \mathcal{L}$ to an output of fixed length in $\{0,1\}^\lambda$. The *PRF-security* of a pseudorandom function PRF measures the ability of an adversary $\mathcal{A}$ to distinguish the output of PRF from random:

$$\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF},\mathcal{A}} = \left|\Pr\left[k \leftarrow\!\!\$\ \mathcal{K} : \mathcal{A}^{\mathsf{PRF}(k,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{R(\cdot)} \Rightarrow 1\right]\right|$$

where $R : \mathcal{L} \to \{0,1\}^\lambda$ is a truly random function.

A pseudorandom function satisfies *dual-PRF-security* [5] if it is a pseudorandom function with respect to either of its inputs k or ℓ being the key, i.e., if both PRF and $\mathsf{PRF}' : (x, y) \mapsto \mathsf{PRF}(y, x)$ have PRF-security.

Definition A.3 (Message authentication code and existential unforgeability under chosen message attack). A *message authentication code* $\mathsf{MAC} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\lambda$ maps a key $k \in \mathcal{K}$ and a message $m \in \{0,1\}^*$ to an authentication tag of fixed length in $\{0,1\}^\lambda$. The *existential unforgeability under chosen message attack* (EUF-CMA) measures the ability to forge an authentication tag on a new message, given access to a tag generation oracle, as shown in Fig. 6: $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}} = \Pr\left[G^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}} \Rightarrow 1\right]$.

$G^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAC},\mathcal{A}}$

1: $k \leftarrow\!\!\$\ \mathcal{K}$
2: $L \leftarrow\!\!\$\ \emptyset$
3: $(m, t) \leftarrow\!\!\$\ \mathcal{A}^{O}$
4: **return** $[\![(t = \mathsf{MAC}(k, m)) \wedge (m \notin L)]\!]$

Oracle $O(z)$

1: $L \leftarrow L \cup \{z\}$
2: **return** $\mathsf{MAC}(k, z)$

Figure 6: Security experiment for existential unforgeability under chosen message attack (EUF-CMA-security) of a message authentication code MAC.

B OW-CCA+ROM ⇒ PRF-IND-CCA

Proof of Theorem 2.2. Let PRF be a random oracle. Let $\mathcal{A}$ be an adversary against the PRF-IND-CCA security of PRF, KEM. We will construct a reduction $\mathcal{B}$ against the OW-CCA security of KEM.

The reduction $\mathcal{B}$ appears in Fig. 7. In Fig. 7, Δ denotes a distinguished internal value that is different from all values the adversary is allowed to query (therefore the outputs of $F(\Delta, \ell)$ will be independent from any $F(\mathsf{ss}, \ell)$ queries the adversary may make).

Let E be the event that $\mathcal{A}$ queries $\mathsf{ss}^\star$ to F. Note that $\mathcal{B}$'s simulation of the PRF-IND-CCA experiment to $\mathcal{A}$ is perfect up until $\mathcal{A}$ queries $\mathsf{ss}^\star$ to F (i.e., up until when event E occurs), after which the simulation may not be correct.

When E does not occur, $\mathcal{A}$ has no information about b and thus $\mathcal{A}$'s advantage in the PRF-IND-CCA game is 0. Consequently, $\Pr[E] \geq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM},\mathcal{A}}$. When E does occur, $\mathsf{ss}^\star$ is in one of the entries in L, so $\mathcal{B}$ has a chance of at least $1/q$ of selecting $\mathsf{ss}^\star$ as ss'. Thus, $\Pr\left[\mathsf{ss}' = \mathsf{ss}^\star | E\right] \geq 1/q$.

Putting these together, we have that

$$\mathsf{Adv}^{\mathsf{OW\text{-}CCA}}_{\mathsf{KEM},\mathcal{B}} \geq \Pr\left[\mathsf{ss}' = \mathsf{ss}^\star | E\right] \Pr[E] \geq \frac{1}{q}\,\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM},\mathcal{A}}$$

which yields the desired result. □

C REDUCTIONIST SECURITY ANALYSIS OF KEMTLS

Our approach adapts the security model and reductionist security analysis of the TLS 1.3 handshake by Dowling, Fischlin, Günther, and Stebila [32, 33] for KEMTLS.

$\underline{\mathcal{B}^{\mathcal{A},\mathcal{O}}(\mathsf{pk}^\star, \mathsf{ct}^\star)}$
1: $L \leftarrow \emptyset$
2: $(\ell^\star, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}',F}(\mathsf{pk}^\star, \mathsf{ct}^\star)$
3: $K \leftarrow\!\!\$\ \{0,1\}^\lambda$
4: $b' \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}',F}(\mathsf{st}, K)$
5: $(\mathsf{ss}', \ell', z') \leftarrow\!\!\$\ L \setminus \{(\Delta, *, *) \in L\}$
6: **return** ss'

$\underline{\text{Oracle } \mathcal{O}'(\mathsf{ct}, \ell)}$
1: **if** $\mathsf{ct} \neq \mathsf{ct}^\star$ **then**
2: $\quad \mathsf{ss} \leftarrow \mathcal{O}(\mathsf{ct})$
3: $\quad$ **return** $F(\mathsf{ss}, \ell)$
4: **else if** $(\mathsf{ct} = \mathsf{ct}^\star) \wedge (\ell \neq \ell^\star)$ **then**
5: $\quad$ **return** $F(\Delta, \ell)$
6: **else**
7: $\quad$ **return** $\perp$

$\underline{\text{Oracle } F(\mathsf{ss}, \ell)}$
1: **if** $\exists z : (\mathsf{ss}, \ell, z) \in L$ **then**
2: $\quad$ **return** z
3: **else**
4: $\quad z \leftarrow\!\!\$\ \{0,1\}^\lambda$
5: $\quad L \leftarrow L \cup \{(\mathsf{ss}, \ell, z)\}$
6: $\quad$ **return** z

Figure 7: Reduction $\mathcal{B}$ for the proof of Theorem 2.2.

C.1 Model syntax

The set $\mathcal{U}$ denotes the set of identities of honest participants in the system. Every identity $S \in \mathcal{U}$ is associated with a certified long-term (server) KEM public key pk_S and corresponding private key sk_S. Participants that only act as clients also possess such a keypair, but do not use it.

Each participant can run multiple instances of the protocol, each of which is called a session. Sessions of a protocol are, for the purposes of modelling, uniquely identified by some administrative label, $\pi \in \mathcal{S} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$, which is a 3-tuple (U, V, n), such that it identifies the nth local session of U talking to intended *partner* V. In a multi-stage protocol, each session consists of multiple stages, run sequentially with shared state; each stage aims to establish a key. Let $\mathsf{M} \in \mathbb{N}$ denote the number of stages. (In KEMTLS, $\mathsf{M} = 6$.)

The security model is parameterized by several characteristics which allow for precise specification of properties of every key established. These are:

- $\mathsf{AUTH} \subseteq \{u_1, \ldots, u_\mathsf{M} \mid u_i \in \{1, \ldots, \mathsf{M}\}\}$. A vector indicating by which stage a key is considered to be authenticated: if $u_i = j$, then, once stage j has accepted, the key established in stage i is considered to be authenticated. Some keys may be considered authenticated right away ($u_i = i$), whereas other keys may only be considered authenticated retroactively ($u_i > i$), after some additional confirmation message has been received.
- $\mathsf{USE} \in \{\text{internal}, \text{external}\}^\mathsf{M}$: USE_i indicates if a stage-i key is used internally in the key exchange protocol. Internally used keys require a little bit of extra care when testing them, while externally used keys may only be used outside the handshake protocol. For KEMTLS, we set this to (internal, internal, internal, internal, external, external).

For each session, each participant maintains the following collection of session-specific information. Many of the values are vectors of length M, with values for each stage.

- $\mathsf{id} \in \mathcal{U}$: the identity of the session *owner*,
- $\mathsf{pid} \in \mathcal{U} \cup \{*\}$: the identity of the *intended* communication partner. This partner may be unknown, which we indicate by the wildcard symbol '$*$'.
- $\mathsf{role} \in \{\text{initiator}, \text{responder}\}$
- $\mathsf{auth} \in \mathsf{AUTH}$: auth_i indicates at what stage $j \geq i$ the unilateral authentication level for stage i is achieved. For unilateral authentication, we set this to (6, 6, 3, 4, 5, 6). Note that the first two keys are only authenticated at the very last stage.
- $\mathsf{st}_\mathsf{exec} \in (\mathsf{ACCEPTED} \cup \mathsf{RUNNING} \cup \{\perp\})$ indicates the state of execution, where $\mathsf{RUNNING} = \{\mathsf{running}_i \mid i \in \mathbb{N}\}$, $\mathsf{ACCEPTED} = \{\mathsf{accepted}_i \mid i \in \mathbb{N}\}$. Sets to $\mathsf{accepted}_i$ when a session accepts the i-th key, and to $\mathsf{running}_i$ when a session continues after accepting. When rejecting a key, this is set to $\perp$ and the protocol does not continue. Initially set to $\mathsf{running}_0$.
- $\mathsf{stage} \in \{0, \ldots, M\}$. The current stage, initialized to 0. It is incremented to i when $\mathsf{st}_\mathsf{exec}$ reaches $\mathsf{accepted}_i$.
- $\mathsf{sid} \in \left(\{0,1\}^* \cup \{\perp\}\right)^\mathsf{M}$: the session identifier in stage i. Initially set to $\perp$, it is updated when reaching acceptance in that stage.
- $\mathsf{cid} \in \left(\{0,1\}^* \cup \{\perp\}\right)^\mathsf{M}$: the contributive identifier in stage i. Initially set to $\perp$ and updated until reaching acceptance in that stage.
- $\mathsf{key} \in \left(\{0,1\}^* \cup \{\perp\}\right)^\mathsf{M}$. Initially $\perp$, key_i indicates the key established in stage i and is set on acceptance.
- $\mathsf{st}_\mathsf{key} \in \{\text{fresh}, \text{revealed}\}^\mathsf{M}$. $\mathsf{st}_{\mathsf{key}_i}$ indicates the state of the session key in stage i; initially set to fresh.
- $\mathsf{tested} \in \{\text{true}, \text{false}\}^\mathsf{M}$. tested_i indicates if key_i has been tested. Initially false.
- $\mathsf{corrupted} \in \{0, \ldots, M, \infty\}$ indicates at what stage a Corrupt was issued to its owner or intended partner. Initially set to ∞, to indicate it has not happened.

For a session identified by π, we may write $\pi.X$ as shorthand to refer to that session's element X.

We define the *partner* of π at stage i to be the π', such that $\pi.\mathsf{sid}_i = \pi'.\mathsf{sid}_i \neq \perp$. Correctness requires that, in a honest joint execution of the protocol, this equality holds for all stages on acceptance.

We say that a session π has been *corrupted* if either π or their intended communication partner $\pi.\mathsf{pid}$ has been corrupted, that is, their secret key $\mathsf{sk}_{\pi.\mathsf{pid}}$ has been compromised.

Upgradable authentication. KEMTLS derives keys in early stages that might only be authenticated at a later stage. The auth vector captures at what stage exactly this happens for each key. However, we need to be careful to consider how corruptions affect this authentication level. We need to disallow stage i to become authenticated at some later stage $j > i$ if the involved parties have been corrupted before stage j accepts. Without this restriction, an adversary might impersonate up to stage i to obtain the unauthenticated secrets, followed by a corruption at stage j to authenticate.

We define the *current authentication level* (what DFGS call "rectified authentication") of π at stage i, $\mathsf{curr\_auth}_i$, as

$$\mathsf{curr\_auth}_i = \begin{cases} \mathsf{unilateral}, & \text{if } \pi.\mathsf{stage} = j \rightarrow (j \geq \mathsf{auth}_i \wedge \pi.\mathsf{corrupted} \geq \mathsf{auth}_i) \\ \mathsf{unauthenticated}, & \text{otherwise.} \end{cases}$$

C.2 Adversary interaction

Following DFGS [32, 33], our two security properties, Match security and Multi-Stage security, take place within the same adversary interaction model. The adversary $\mathcal{A}$ is a probabilistic algorithm which controls the communication between all parties and thus can intercept, inject or drop any message. In this type of model, even two honest parties require $\mathcal{A}$ to facilitate communication to establish a session.

Some combinations of queries will be restricted; for example, allowing the adversary to both reveal and test a particular session key would allow the adversary to trivially win the test challenge in Multi-Stage security, and thus does not model security appropriately. We will record if the adversary trivially loses with the flag lost (initially set to false).

The first two queries the adversary has access to model honest protocol functionality:

- NewSession(U, V, role): Creates a new session π where $\pi.\mathsf{id} = U$ will be intending to partner with $\mathsf{pid} = V$ in role. V may be left unspecified ($V = *$). We add π to the list of sessions.
- Send(π, m): Sends message m to the session π. If π is not in the list of sessions, we return $\perp$. Otherwise, Send runs the protocol on behalf of $\pi.\mathsf{id}$. It will record the updated state, and return both the response message and $\pi.\mathsf{st}_{\mathsf{exec}}$. To allow us to let $\pi.\mathsf{id}$ initiate the protocol, we may submit the special symbol $m = \mathsf{init}$ if $\pi.\mathsf{role} = \mathsf{initiator}$.
  The adversary may not test any keys that have already been used. Because internal keys may be used immediately, we will pause execution whenever any key is accepted. Send will then return $\mathsf{accepted}_i$ to the adversary. Whenever the adversary decides to continue this session, they may submit Send(π, continue). This will continue the protocol as specified, and the adversary will obtain the next protocol message and resulting $\mathsf{st}_{\mathsf{exec}}$.
  Additionally, whenever $\pi.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$ is reached for some stage i, we need to ensure that the session is consistent with partnered sessions. If there is a partnered session $\pi' \neq \pi$, with $\pi'.\mathsf{tested}_i = \mathsf{true}$, then we set $\pi.\mathsf{tested}_i \leftarrow \mathsf{true}$. For internal keys ($\mathsf{USE}_i = \mathsf{internal}$), we also set $\pi'.\mathsf{key}_i \leftarrow \pi.\mathsf{key}_i$ to ensure session keys are used consistently.
  Finally, if we reach $\pi.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$ for some i and $\pi.\mathsf{corrupted} = \mathsf{true}$, we set $\pi.\mathsf{st}_{\mathsf{key}_i} \leftarrow \mathsf{revealed}$.

The next two queries model the adversary's ability to compromise participants and learn some secret information:

- Reveal(π, i): Reveals the session key $\pi.\mathsf{key}_i$ to the adversary. If there is no such session π in the list of sessions, or if $\pi.\mathsf{stage} < i$, return $\perp$. Else, we set $\pi.\mathsf{st}_{\mathsf{key}_i} \leftarrow \mathsf{revealed}$ and return $\pi.\mathsf{key}_i$.
- Corrupt(U): Provide the adversary with the long-term secret key sk_U of U. We record the time of corruption, if not already set, in all sessions π where $\pi.\mathsf{id} = U$ or $\mathsf{pid} = U$, by setting $\pi.\mathsf{corrupted} = \pi.\mathsf{stage}$.
  In addition, in all sessions with $\pi.\mathsf{id} = U$ or $\mathsf{pid} = U$, we record all keys $\pi.\mathsf{st}_{\mathsf{key}_i} \leftarrow \mathsf{revealed}$ if $i > \pi.\mathsf{stage}$. This models that all keys that will be established after this stage have been potentially disclosed, but any prior keys remain fresh. This models forward secrecy, which in our protocol starts from stage 1.

The final query models the challenge to the adversary of breaking a key that was established by honest parties:

- Test(π, i) Tests the session key $\pi.\mathsf{key}_i$.
  We require the session to be honestly started, so it must be in the list of sessions. We also require that $\pi.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$ and the key must not have been tested before ($\pi.\mathsf{tested}_i \neq \mathsf{true}$). If these requirements do not hold, we return $\perp$.
  If the session key in stage i is internal ($\mathsf{USE}_i = \mathsf{internal}$), we require a partnered session π' in the list of sessions to also have $\pi'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$. Otherwise we set $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that any partnered session has also not yet used the key.
  To test unauthenticated stages, we require having an honest, contributive partner. Otherwise the adversary may trivially win through impersonation. That is why, if $\pi.\mathsf{curr\_auth}_i = \mathsf{unauthenticated}$ and there is no $\pi' \neq \pi$ in the list of sessions where $\pi'.\mathsf{cid}_i = \pi.\mathsf{cid}_i$, we set $\mathsf{lost} \leftarrow \mathsf{true}$.
  The Test oracle has a uniformly random bit b, which is fixed throughout the game. Set $\pi.\mathsf{tested}_i \leftarrow \mathsf{true}$. If test bit $b = 0$, we sample a key $K \leftarrow\!\!\$\, \mathcal{K}$, from $\mathcal{K}$ the set of all possible session keys. If test bit $b = 1$, we set $K \leftarrow \pi.\mathsf{key}_i$. To make sure that the selected K is consistent with any later used keys, we set $\pi.\mathsf{key}_i = K$ if $\mathsf{USE}_i = \mathsf{internal}$.
  We then ensure consistency with any partnered sessions: where $\pi.\mathsf{sid}_i = \pi'.\mathsf{sid}_i$ and $\pi.\mathsf{st}_{\mathsf{exec}} = \pi'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$, set $\pi'.\mathsf{tested}_i \leftarrow \mathsf{true}$, and, if $\mathsf{USE}_i = \mathsf{internal}$, set $\pi'.\mathsf{key}_i \leftarrow K$.
  Return K to the adversary.

C.3 Specifics of KEMTLS in the model

For the proofs in the subsequent subsections, KEMTLS is as specified in Fig. 5. To fully instantiate the model, the model parameters auth and USE are fixed as indicated above. We additionally need to state what the session identifiers sid_i and contributive identifiers cid_i are for all stages in the protocol.

Whenever a stage is accepted, its session identifier is set to consist of a label and all handshake messages up to that point:

$\mathsf{sid}_1 = (\text{"CHTS"}, \texttt{ClientHello} \ldots \texttt{ServerHello})$,

$\mathsf{sid}_2 = (\text{"SHTS"}, \texttt{ClientHello} \ldots \texttt{ServerHello})$,

$\mathsf{sid}_3 = (\text{"CAHTS"}, \texttt{ClientHello} \ldots \texttt{ClientKemCiphertext})$,

$\mathsf{sid}_4 = (\text{"SAHTS"}, \texttt{ClientHello} \ldots \texttt{ClientKemCiphertext})$,

$\mathsf{sid}_5 = (\text{"CATS"}, \texttt{ClientHello} \ldots \texttt{ClientFinished})$,

$\mathsf{sid}_6 = (\text{"SATS"}, \texttt{ClientHello} \ldots \texttt{ServerFinished})$.

For the contributive identifiers cid_i we need special care for the first stage. In stage $i = 1$, the client and server set, upon sending (client) or receiving (server) the `ClientHello` message,

cid_1 = ("CHTS", `ClientHello`). When they next send (server) or receive (client) the `ServerHello` response, they update this to $\mathsf{cid}_1 = \mathsf{sid}_1$. All other contributive identifiers are set to $\mathsf{cid}_i = \mathsf{sid}_i$ whenever sid_i is set.

C.4 Match security

Match security models session matching: it ensures that the session identifier $\pi.\mathsf{sid}$ matches the partnered $\pi'.\mathsf{sid}$. (Splitting AKE security into two distinct properties, Match security and session key indistinguishability, is the approach of [23, 24, 32, 33, 35].)

Definition C.1. Match Security

Let KE be a M-stage key exchange protocol with properties (AUTH, USE), and let $\mathcal{A}$ be a probabilistic adversary interacting with KE via the queries defined in Appendix C.2. $\mathcal{A}$ tries to win the following game $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$:

Setup The challenger generates long-term public and private key pairs $(\mathsf{pk}_U, \mathsf{sk}_U)$ for each participant $U \in \mathcal{U}$. These are provided to $\mathcal{A}$.

Query The adversary has access to the queries NewSession, Send, Reveal, Corrupt, and Test.

Stop At some point, the adversary stops with no output.

Let π, π' be distinct partnered sessions with some stage $i \in \{1, \ldots, \mathsf{M}\}$ for which $\pi.\mathsf{sid}_i = \pi'.\mathsf{sid}_i \neq \bot$.

We say that $\mathcal{A}$ wins $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$ if it can falsify any one of the following conditions:

(1) π, π' agree on the same key at every stage i, i.e., they must have $\pi.\mathsf{key}_i = \pi'.\mathsf{key}_i$.
(2) π, π' have opposite roles: $\pi.\mathsf{role} \neq \pi'.\mathsf{role}$.
(3) π, π' agree on intended authentication types at any stage i, i.e., $\pi.\mathsf{auth}_i = \pi'.\mathsf{auth}_i$.
(4) π, π', partnered in some stage i, have set and agree on the contributive identifier, i.e., $\pi.\mathsf{cid}_i = \pi'.\mathsf{cid}_i \neq \bot$.
(5) π, π', if they reached the unilateral authentication level, agree that the id of the responder is the pid of the initiator, I.e., at any stage i for which $\pi.\mathsf{auth}_i = \pi'.\mathsf{auth}_i \leq \pi.\mathsf{stage}$, $\pi.\mathsf{role} = \mathsf{initiator}$ and $\pi'.\mathsf{role} = \mathsf{responder}$ implies $\pi.\mathsf{pid} = \pi'.\mathsf{id}$.
(6) π, π' (not necessarily distinct) have distinct session identifiers across distinct stages i, j: $\pi.\mathsf{sid}_i = \pi'.\mathsf{sid}_j$ implies $i = j$.
(7) π, π' do not have any third partner session π'', i.e., at any stage i, having $\pi.\mathsf{sid}_i = \pi'.\mathsf{sid}_i = \pi''.\mathsf{sid}_i \neq \bot$ implies $\pi = \pi'$, $\pi' = \pi''$, or $\pi = \pi''$.

Theorem C.2. KEMTLS *is* Match*-secure. In particular, for any adversary $\mathcal{A}$, An efficient adversary $\mathcal{A}$ has advantage*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KEMTLS},\mathcal{A}} \leq n_s^2 / 2^{|nonce|},$$

where n_s is the number of sessions and $|nonce|$ the length of the nonce in bits.

Proof. We need to show each property of Match security (Definition C.1) holds:

(1) The session identifiers are defined to contain all handshake messages. KEM messages and hashes of those messages are only inputs into the key schedule. In stages 1 and 2, the input to the agreed keys is the ephemeral KEM shared secret and the messages up to `ServerHello`. For stage 3 and 4, the input to the agreed keys are the previous keys, messages up to `ClientKemCiphertext` and the static KEM shared secret. For the final stages 5 and 6, the input to the keys is the previous keys and the messages up to `ClientFinished` and `ServerFinished` respectively. It is easy to confirm that this is all included in the session identifiers, and that sharing session identifiers is indeed required to derive the same keys.
(2) No initiator or responder will ever accept a wrong-role incoming message, so any pair of two sessions must have both an initiator and a responder. We will later show that at most two sessions have the same sid, implying that this pairing will be unique and thus opposite.
(3) By definition, KEMTLS has a fixed authentication vector that is the same for all sessions.
(4) By definition, cid_i is final and equal to sid_i whenever stage i is accepted.
(5) The partnered sessions only have to agree once they reach the unilateral authentication level. The identity is learned through the `ServerCertificate` sent by the responder. Because Match security only concerns honest sessions, the `ServerCertificate` received by the initiator will set the correct pid.
(6) Every stage's session identifier is defined to have a unique label, thus there can be no confusion across distinct stages.
(7) The session identifiers include the random nonce and KEM public key and ciphertext. For three sessions to have the same identifier, we would need to have a collision of these values picked by honest servers and clients. Without making assumptions on the KEM scheme, we can rely on distinctness of nonces under the birthday bound on n_s the number of sessions: the probability of failing in n_s sessions is less than $n_s^2/2^{|nonce|}$, which is negligible in the bit-length of the nonce.

□

C.5 Multi-Stage security

Secrecy of each stage key is defined as being indistinguishable from a random key, Bellare–Rogaway-style [7]. The Multi-Stage experiment was introduced by [35] and was also used by DFGS for TLS 1.3 [32, 33].

Definition C.3. Multi-Stage security

Let KE be an M-stage key exchange protocol with properties (AUTH, USE), and let $\mathcal{A}$ be a probabilistic adversary interacting with KE via the queries defined in Appendix C.2. The adversary tries to win the following game $G^{\mathsf{Multi\text{-}Stage}}_{\mathsf{KE},\mathcal{A}}$:

Setup The challenger generates all long-term keys $(\mathsf{pk}_U, \mathsf{sk}_U)$ for all identities $U \in \mathcal{U}$, picks a uniformly random bit b (for the Test queries), and initializes lost $\leftarrow$ false. The public keys are provided to $\mathcal{A}$.

Query The adversary has access to the queries NewSession, Send, Reveal, Corrupt, and Test.

Stop At some point, $\mathcal{A}$ stops and outputs their guess b' of b.

Finalize The challenger sets lost $\leftarrow$ true if there exist two sessions π, π' where at some stage i, $\pi.\mathsf{sid}_i = \pi'.\mathsf{sid}_i$, $\pi.\mathsf{st}_{\mathsf{key}_i} =$ revealed and $\pi'.\mathsf{tested} =$ true. (This prevents the adversary

from revealing and testing the same key of some stage in a (partnered) session.) Note that Test may also set lost = true, which concerns authentication properties.

$\mathcal{A}$ wins the game if they output the correct $b' = b$ and lost = false, in which case $G^{\text{Multi-Stage}}_{\text{KE},\mathcal{A}}$ outputs 1. If lost = true or $b' \neq b$, the output of the game is a bit sampled uniformly from $\{0,1\}$. (Note that authentication properties of KE are captured by the setting of lost in the definitions of queries, e.g., Corrupt.)

The Multi-Stage-advantage of $\mathcal{A}$ for properties (AUTH, USE) is defined as:

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

Theorem C.4. *Let $\mathcal{A}$ be an algorithm, and let n_s be the number of sessions and n_u be the number of parties. There exist algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_{14}$, described in the proof, such that*

$$\mathsf{Adv}^{\text{Multi-Stage}}_{\text{KEMTLS},\mathcal{A}} \leq 6n_s \left(\begin{array}{l} \mathsf{Adv}^{\text{COLL}}_{\text{H},\mathcal{B}_1} \\ +n_u \left(\begin{array}{l} \mathsf{Adv}^{\text{PRF-IND-CCA}}_{F,\mathcal{B}_2} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Extract},\mathcal{B}_3} \\ + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_4} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_5} \\ + \mathsf{Adv}^{\text{EUF-CMA}}_{\text{HMAC},\mathcal{B}_6} \end{array} \right) \\ +n_s \left(\begin{array}{l} \mathsf{Adv}^{\text{IND-CPA}}_{\text{KEM},\mathcal{B}_7} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Extract},\mathcal{B}_8} \\ + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_9} + \mathsf{Adv}^{\text{dual-PRF-sec}}_{\text{HKDF.Extract},\mathcal{B}_{10}} \\ + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_{11}} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_{12}} \\ + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Extract},\mathcal{B}_{13}} + \mathsf{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_{14}} \end{array} \right) \end{array} \right)$$

where F is a function that combines KEM *and* HKDF *as follows:*

$$F(\mathsf{ct}, lbl, tr) = \mathsf{HKDF.Expand}(\,\mathsf{HKDF.Extract}(lbl, \mathsf{KEM.Decapsulate}(\mathsf{ct})), \texttt{"derived"}, tr)\ .$$

Proof. The proof follows the basic structure of the proof of DFGS [32, 33] for the TLS 1.3 signed-Diffie–Hellman full handshake, and proceeds by a sequence of games. During the game hops, we will often replace a key in a derivation step by a value replaced in a previous game; whenever this happens, we replace it both in the target session π, and in any partnered sessions. In the following, whenever we say we replace a value by a random one, we mean that we replace it by a uniformly randomly chosen bitstring of the same (fixed) length.

Game 0 We define G_0 to be the original Multi-Stage game:

$$\mathsf{Adv}^{\text{Multi-Stage}}_{\text{KEMTLS},\mathcal{A}} = \mathsf{Adv}^{G_0}_{\mathcal{A}}\ .$$

Game 1 We now restrict $\mathcal{A}$ to only make a single Test query. This reduces its advantage by at most $1/6n_s$ for the six stages, based on a hybrid argument by Günther [41]. Any single-query adversary $\mathcal{A}_1$ can emulate a multi-query adversary $\mathcal{A}$ by guessing a to-be-tested session in advance. For any other Test queries $\mathcal{A}$ may submit, $\mathcal{A}_1$ can substitute by Reveal queries. $\mathcal{A}_1$ will need to know how sessions are partnered. Early partnering is decided by public information $(\mathsf{sid}_1, \mathsf{sid}_2)$, but later sids are encrypted. However, $\mathcal{A}_1$ can just reveal the handshake traffic keys to decrypt the subsequent information.

We get the following advantage by letting transformed adversary $\mathcal{A}_1$ guess the right session:

$$\mathsf{Adv}^{G_0}_{\mathcal{A}} \leq 6n_s \cdot \mathsf{Adv}^{G_1}_{\mathcal{A}_1}\ .$$

With this transformation, we can now refer to *the* session π tested at stage i, and assume that we know the tested session π at the outset.

Game 2 In this game, the challenger will abort if any two honest sessions compute the same hash for different inputs of hash function H. If this happens, it induces a reduction $\mathcal{B}_1$ that can break the collision-resistance of H. If a collision occurs, $\mathcal{B}_1$ outputs the two distinct input values. Thus:

$$\mathsf{Adv}^{G_1}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_2}_{\mathcal{A}_1} + \mathsf{Adv}^{\text{COLL}}_{\text{H},\mathcal{B}_1}\ .$$

We now need to consider two separate cases of G_2:

A. (denoted G_A) The tested session π has no honest, contributive partner in stage 1. Formally, there exists no $\pi' \neq \pi$ where $\pi'.\mathsf{cid}_1 = \pi.\mathsf{cid}_1$.
B. (denoted G_B) The tested session π *does* have an honest, contributive partner.

We will consider the advantage of the adversary separately for these two cases:

$$\mathsf{Adv}^{G_2}_{\mathcal{A}_1} \leq \max\{\mathsf{Adv}^{G_A}_{\mathcal{A}_1}, \mathsf{Adv}^{G_B}_{\mathcal{A}_1}\} \leq \mathsf{Adv}^{G_A}_{\mathcal{A}_1} + \mathsf{Adv}^{G_B}_{\mathcal{A}_1}$$

Case A. Test without a partner

In this case, there is no honest, contributive partner in stage 1. This implies there is no such partner at any stage and that $\mathcal{A}_1$ is impersonating such a partner to the tested session.

By definition of Test an adversary can not win if it is issued to an initiator session at some stage i where $\mathsf{curr\_auth}_i$ = unauthenticated. The stage 1 and 2 keys are only (unilaterally) authenticated after π received the `ServerFinished` message and stage 6 is reached. Otherwise, $\mathcal{A}_1$ might just impersonate the other party and the key material of stages 1 and 2 could be trivially obtained. If $\mathcal{A}_1$ impersonates the initiator, they set ephemeral $\mathsf{sk}_e \leftarrow sk_{\mathcal{A}}$ and trivially decapsulate the key, that π encapsulated to $\mathsf{pk}_{\mathcal{A}}$. Or they are impersonating responder and trivially obtain the key from the KEM.Encapsulate operation on π's ephemeral pk_e. For the same reasons we do not allow the responder to be tested, because the client is never authenticated.

The keys from stages 3–6 are at first implicitly authenticated, because their derivation involves the responder's long-term sk.

Note that the $\mathsf{curr\_auth}_i$ = unilateral property also requires the impersonated partner has not been corrupted before authentication is reached at stage 3.

Game A1 In this game, the challenger guesses the peer identity $U \in \mathcal{U}$ of the tested π. We abort the game if the guess was incorrect. $\mathcal{A}_1$'s advantage is reduced by at most a factor of the number of users n_u:

$$\mathsf{Adv}^{G_A}_{\mathcal{A}_1} \leq n_u \cdot \mathsf{Adv}^{G_{A1}}_{\mathcal{A}_1}\ .$$

Game A2 In this game, in the tested session, we replace the secret dAHS by a uniformly random $\widetilde{\text{dAHS}}$. Subsequently, the game A2 challenger modifies all derivations that stem

from the now-replaced key $\widetilde{\text{dAHS}}$ to use the randomized value. This updates CAHTS, SAHTS and MS.
We can use any $\mathcal{A}_1$ that can distinguish this replacement to construct a PRF-IND-CCA adversary $\mathcal{B}_2$.
The PRF in the PRF-IND-CCA experiment is built from two applications of HKDF as follows: $F(\text{ct}, lbl, tr)$ = HKDF.Expand(HKDF.Extract (lbl, KEM.Decapsulate(ct)), "derived", tr). Note that we split the PRF label into lbl and tr components to represent the HKDF.Extract label and the HKDF.Expand transcript.
$\mathcal{B}_2$ simulates Game A2 for $\mathcal{A}_1$ as follows. $\mathcal{B}_2$, who for F and KEM is a PRF-IND-CCA adversary, receives as input a KEM public key $\text{pk}^\star$. It sets it as the static public key of the server U it guessed in game A1 as the peer of the tested session. For all sessions except the session guessed as the test session in game G_1, $\mathcal{B}_2$ relies on its decapsulation oracle to successfully simulate U's decapsulation operations and corresponding calculations of dAHS. For the test session at the client, $\mathcal{B}_2$ uses the challenge ciphertext $\text{ct}^\star$ and the challenge shared secret $\text{ss}^\star$, with label lbl = dHS and tr = CH...CKC. Note that $\mathcal{B}_2$ never queries its PRF-IND-CCA oracle on the prohibited query ($\text{ct}^\star$, lbl = dHS, tr = CH...CKC) since the (a) the peer session does not exist and (b) no two honest sessions have the same contributive identifier cid_3 due to Match-security, which is the transcript used in this query.
When the PRF-IND-CCA challenger was using the real shared secret (its bit $b = 0$), $\mathcal{B}_2$ perfectly simulates game A1 to $\mathcal{A}_1$. When the PRF-IND-CCA challenger was using a random shared secret (its bit $b = 1$), $\mathcal{B}_2$ perfectly simulates game A2 to $\mathcal{A}_1$. Thus:

$$\text{Adv}^{G_{A1}}_{\mathcal{A}_1} \leq \text{Adv}^{G_{A2}}_{\mathcal{A}_1} + \text{Adv}^{\text{PRF-IND-CCA}}_{F,\mathcal{B}_2}$$

Game A3 In this game, in the tested session, we replace CAHTS, SAHTS and MS by uniformly random $\widetilde{\text{MS}}$, $\widetilde{\text{CAHTS}}$ and $\widetilde{\text{SAHTS}}$. Subsequently, the challenger modifies all derivations that stem from the now-replaced key $\widetilde{\text{MS}}$ to use the randomized value. This affects finished keys fk_c, fk_s and transport keys CATS, SATS.
We can use any $\mathcal{A}_1$ that can distinguish the replacement of MS by $\widetilde{\text{MS}}$ to construct a reduction $\mathcal{B}_3$ from the PRF-sec security of HKDF.Extract.
Furthermore, an $\mathcal{A}$ that can distinguish the replacement of CAHTS and SAHTS by $\widetilde{\text{CAHTS}}$ and $\widetilde{\text{SAHTS}}$, respectively, implies that we can construct an adversary $\mathcal{B}_4$ to the PRF-sec security of HKDF.Expand.
Together, these yield:

$$\text{Adv}^{G_{A2}}_{\mathcal{A}_1} \leq \text{Adv}^{G_{A3}}_{\mathcal{A}_1} + \text{Adv}^{\text{PRF-sec}}_{\text{HKDF.Extract},\mathcal{B}_3} + \text{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_4}.$$

This yields key indistinguishability for the stage-3 and stage-4 keys CAHTS and SAHTS.

Game A4 In this game, in the tested session, we replace finished keys fk_c, fk_s, and application-data transport keys CATS and SATS by uniformly random $\widetilde{\text{fk}_c}$, $\widetilde{\text{fk}_s}$, $\widetilde{\text{CATS}}$ and $\widetilde{\text{SATS}}$. Subsequently, the corresponding evaluations of HMAC use the now randomized values $\widetilde{\text{fk}_c}$, $\widetilde{\text{fk}_s}$.
We can use any adversary that can distinguish any of these replacements to construct a reduction $\mathcal{B}_5$ to the PRF-sec security of HKDF.Expand, thus yielding:

$$\text{Adv}^{G_{A3}}_{\mathcal{A}_1} \leq \text{Adv}^{G_{A4}}_{\mathcal{A}_1} + \text{Adv}^{\text{PRF-sec}}_{\text{HKDF.Expand},\mathcal{B}_5}.$$

This yields key indistinguishability for the stage-5 and stage-6 keys CATS and SATS.

Game A5 In this game, in the tested session, the challenger aborts the game once the tested session π receives a valid ServerFinished message that has not been produced by any honest session of U. The authentication tag in ServerFinished covers the full transcript and, since Game 2 ruled out hash collisions and Match security ensures unique session identifiers, this transcript is unique across honest sessions.
We can relate the probability of this game aborting to the advantage of $\mathcal{B}_6$ against the EUF-CMA security of HMAC. Formally, we provide reduction $\mathcal{B}_6$ with an HMAC oracle for π.pid = U. We also give it U's long-term public key pk_U. $\mathcal{B}_6$ computes all long-term keys, except U's, and it simulates Game A5 for $\mathcal{A}_1$. Whenever it needs to compute the authentication tag for U in the test session, we use the HMAC oracle. When π receives a valid tag t (from $\mathcal{A}_1$) on the transcript m in the ServerFinished message, $\mathcal{B}_6$ outputs (m, t) as its forgery. Because no other honest session has the transcript m, $\mathcal{B}_6$ has never queried its HMAC oracle on m, and then (m, t) is a winning forgery in the EUF-CMA experiment for HMAC. Thus:

$$\text{Adv}^{G_{A4}}_{\mathcal{A}_1} \leq \text{Adv}^{G_{A5}}_{\mathcal{A}_1} + \text{Adv}^{\text{EUF-CMA}}_{\text{HMAC},\mathcal{B}_6}.$$

Because in case A π has no honest, contributive partner, it follows that the adversary needs to forge ServerFinished to cause the tested session to accept and continue. Without a valid ServerFinished, the remaining unauthenticated keys of the first stages can not be tested. If the challenger does not abort, then $\mathcal{A}_1$'s remaining advantage in predicting b is thus $\text{Adv}^{G_{A5}}_{\mathcal{A}_1} = 0$.

Case B. Test with a partner

Game B1 In this game, the challenger tries to guess the $\pi' \neq \pi$ that is the honest contributive partner to π, i.e., $\pi.\text{cid}_1 = \pi'.\text{cid}_1$.
This reduces the advantage of $\mathcal{A}_1$ by a factor of the number of sessions n_s:

$$\text{Adv}^{G_B}_{\mathcal{A}_1} \leq n_s \cdot \text{Adv}^{G_{B1}}_{\mathcal{A}_1}.$$

Game B2 In this game, we replace the ephemeral secret ss_e derived in sessions π, π' with an uniformly random $\widetilde{\text{ss}_e}$. Subsequently, the challenger modifies all derivations in π and π' from the now-replaced key $\widetilde{\text{ss}_e}$ to use the randomized value, in particular, updating the computation of HS.
Any adversary $\mathcal{A}_1$ that can detect this change can be turned into an adversary $\mathcal{B}_7$ against the IND-CPA security of KEM as follows.
$\mathcal{B}_7$ obtains the IND-CPA challenge $\text{pk}^\star$, $\text{ct}^\star$ and challenge shared secret $\text{ss}^\star$. It simulates π and π' for $\mathcal{A}_1$ by sending the public key $\text{pk}^\star$ in a ClientHello and the ciphertext $\text{ct}^\star$

in the ServerHello reply. It then gives $\mathcal{A}_1$ the challenge shared secret $ss^\star$. $\mathcal{A}_1$ will then give $\mathcal{B}_7$ its guess of $b = 0$ or $b = 1$. If $ss^\star$ was the real shared secret, then $\mathcal{B}_7$ has exactly simulated the G_{B1} to $\mathcal{A}_1$; if $ss^\star$ was a random value, then $\mathcal{B}_7$ has exactly simulated G_{B2} to $\mathcal{A}_1$. Thus:

$$\mathsf{Adv}^{G_{B1}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B2}}_{\mathcal{A}_1} + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM},\mathcal{B}_7}.$$

Game B3 In this game, in the tested session π and its partner π' we replace HS by a uniformly random $\widetilde{\mathrm{HS}}$. Subsequently, the challenger modifies all derivations from the now-replaced key $\widetilde{\mathrm{HS}}$ to use the randomized value. Specifically we update SHTS, CHTS and dHS in π and π to be derived from the new value.
Any adversary $\mathcal{A}_1$ who can tell that HS has been replaced implies the existence of a reduction $\mathcal{B}_8$ that breaks PRF-sec of HKDF.Extract:

$$\mathsf{Adv}^{G_{B2}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B3}}_{\mathcal{A}_1} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_8}.$$

Game B4 In this game, in the tested session π and its partner π', we replace SHTS, CHTS, and dHS, by uniformly random $\widetilde{SHTS}$, $\widetilde{\mathrm{CHTS}}$, and $\widetilde{\mathrm{dHS}}$. Subsequently, the challenger modifies all derivations from the now-replaced key $\widetilde{\mathrm{dHS}}$ to use the randomized value. Specifically we update AHS in π and π to be derived from the new value.
Any adversary $\mathcal{A}_1$ who can tell that SHTS, CHTS or dHS have been replaced implies the existence of a reduction $\mathcal{B}_9$ that breaks PRF-sec of HKDF.Expand:

$$\mathsf{Adv}^{G_{B3}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B4}}_{\mathcal{A}_1} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_9}.$$

This yields key indistinguishability for the stage-1 and stage-2 keys CHTS and SHTS.

Game B5 In this game, in the tested session π and its partner π', we replace AHS by a uniformly random $\widetilde{\mathrm{AHS}}$. Subsequently, the challenger modifies all derivations from the now-replaced key $\widetilde{\mathrm{AHS}}$ to use the randomized value. Specifically we update dAHS in π and π to be derived from the new value.
Note that the computation $\mathrm{dAHS} \leftarrow \mathsf{HKDF.Extract}(\mathrm{AHS}, ss_s)$ places AHS in the "salt" argument of HKDF.Extract, rather than the "input keying material" argument, and thus we invoke the dual-PRF-security of HKDF.Extract.
Any adversary $\mathcal{A}_1$ who can tell that AHS has been replaced implies the existence of a reduction $\mathcal{B}_{10}$ that breaks dual-PRF-sec of HKDF.Extract:

$$\mathsf{Adv}^{G_{B4}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B5}}_{\mathcal{A}_1} + \mathsf{Adv}^{\mathsf{dual\text{-}PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_{10}}.$$

Game B6 In this game, in the tested session π and its partner π', we replace dAHS by a uniformly random $\widetilde{\mathrm{dAHS}}$. Subsequently, the challenger modifies all derivations from the now-replaced key $\widetilde{\mathrm{dAHS}}$ to use the randomized value. Specifically we update SAHTS, CAHTS and MS in π and π to be derived from the new value.
Any adversary $\mathcal{A}_1$ who can tell that dAHS has been replaced implies the existence of a reduction $\mathcal{B}_{11}$ that breaks PRF-sec of HKDF.Expand:

$$\mathsf{Adv}^{G_{B5}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B6}}_{\mathcal{A}_1} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{11}}.$$

Game B7 In this game, in the tested session π and its partner π', we replace SAHTS, CAHTS by uniformly random $\widetilde{\mathrm{CAHTS}}$, $\widetilde{\mathrm{SAHTS}}$. Any adversary $\mathcal{A}_1$ who can tell that dAHS has been replaced implies the existence of a reduction $\mathcal{B}_{12}$ that breaks PRF-sec of HKDF.Expand:

$$\mathsf{Adv}^{G_{B6}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B7}}_{\mathcal{A}_1} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{12}}.$$

This yields key indistinguishability for the stage-3 and stage-4 keys CAHTS and SAHTS.

Game B8 In this game, in the tested session π and its partner π', we replace MS by a uniformly random $\widetilde{\mathrm{MS}}$. Subsequently, the challenger modifies all derivations from the now-replaced key $\widetilde{\mathrm{MS}}$ to use the randomized value. Specifically we update SATS, CATS, fk_s and fk_c in π and π to be derived from the new value.
Any adversary $\mathcal{A}_1$ who can tell that MS has been replaced implies the existence of a reduction $\mathcal{B}_{13}$ that breaks PRF-sec of HKDF.Extract:

$$\mathsf{Adv}^{G_{B7}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B8}}_{\mathcal{A}_1} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Extract},\mathcal{B}_{13}}.$$

Game B9 In this game, in the tested session π and its partner π', we replace SATS, CATS, fk_s and fk_c by uniformly random $\widetilde{\mathrm{SATS}}$, $\widetilde{\mathrm{CATS}}$, $\widetilde{\mathsf{fk}_s}$ and $\widetilde{\mathsf{fk}_c}$.
Any adversary $\mathcal{A}_1$ who can detect these replacements implies the existence of a reduction $\mathcal{B}_{14}$ that breaks PRF-sec of HKDF.Expand:

$$\mathsf{Adv}^{G_{B8}}_{\mathcal{A}_1} \leq \mathsf{Adv}^{G_{B9}}_{\mathcal{A}_1} + \mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{HKDF.Expand},\mathcal{B}_{14}}.$$

This yields key indistinguishability for the stage-5 and stage-6 keys CATS and SATS.
We have now replaced all stages' keys in π by uniformly random, independent values that are only shared between π and π'. The adversary can not reveal them, so the adversary has no information about them. In conclusion, there remains no advantage for the adversary:

$$\mathsf{Adv}^{G_{B9}}_{\mathcal{A}_1} = 0.$$

□

D CLIENT-AUTHENTICATION IN KEMTLS

Although perhaps not used much for web browsing, client authentication is an important optional feature of the TLS handshake. In TLS 1.3 a server can send the client a CertificateRequest message. The client replies with its certificate in a ClientCertificate message and a ClientCertificateVerify message containing a signature. This allows mutual authentication.

In this section, we show how to extend KEMTLS to provide client authentication. Fig. 8 adds a client authentication message flow to KEMTLS.

Recall that we assume that the client does not have the server's certificate when initiating the handshake, and similarly the server does not have the client's certificate in advance. There may be more efficient message flows possible if this is the case, which we leave as future work.

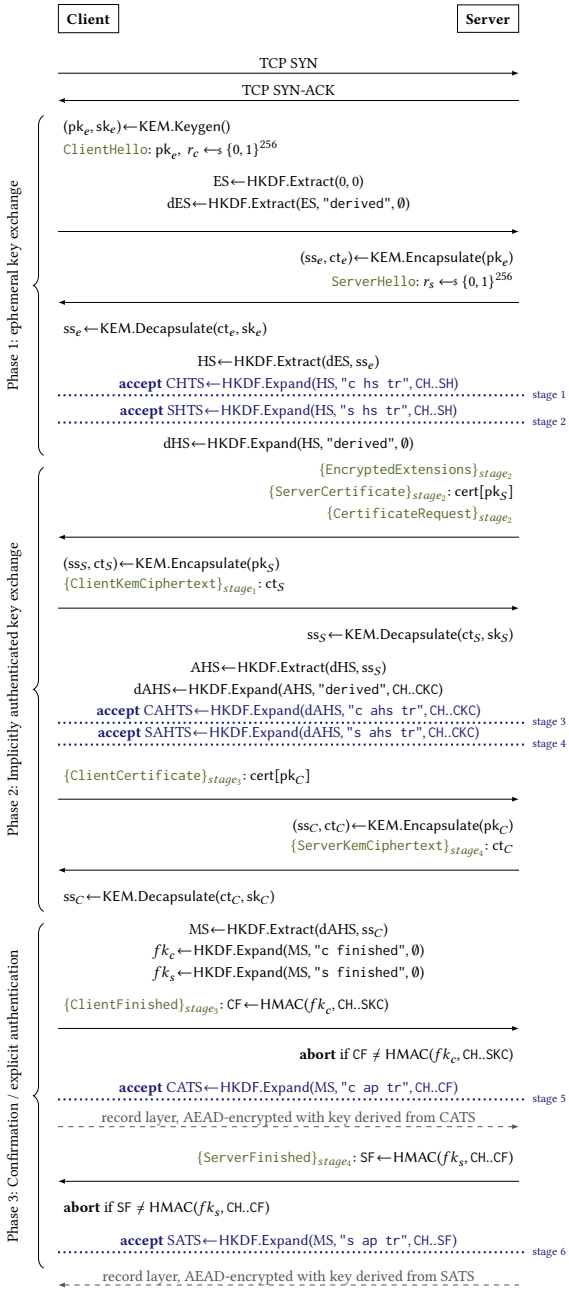

Figure 8: The KEMTLS handshake with client authentication

D.1 Extending KEMTLS with client authentication

In TLS 1.3, a server is only allowed to send a `CertificateRequest` message if it has been authenticated with a certificate [77, Sec. 4.3.2]. This restriction ensures that the certificate containing the identity of the client is only revealed to the intended server. Transferring this property to KEMTLS requires a careful modification of the key schedule. In the KEMTLS key schedule, we derive the CAHTS and SAHTS "authenticated" handshake traffic secrets from the shared secret ss_S encapsulated against the public key in the server's certificate. This allows the client to encrypt its certificate such that it can only be decrypted by someone holding the server certificate's private key.

After that, the server encapsulates against the public key contained in the client certificate to compute another shared secret ss_C. We mix this shared secret ss_C into the derivation of MS (in a straightforward extension of the key schedule of KEMTLS). Mixing together ss_C and ss_S ensures that all application traffic encrypted under keys derived from MS will only be legible to the authenticated server and client; the ephemeral shared secret ss_e further provides forward secrecy.

To prove security of client-authenticated KEMTLS, we would extend our security model (Appendix C) with mutual authentication, which is present in the model of DFGS [32, 33]. At stage 5, client authentication would be achieved. Proving this would follow the same approach as Game A2 (Appendix C.5), using the PRF-IND-CCA property of the KEM and HKDF. If the KEM is appropriately secure, only the intended client should be able to decapsulate and recover ss_C. Thus, ss_C and the MS value and other keys derived from it, are implicitly authenticated keys that the adversary should not be able to compute.

Finally, by sending the `ClientFinished` message containing a MAC under a key derived from MS, the client explicitly authenticates itself to the server.

D.2 Alternative protocol flows

The extension sketched in this section introduces an extra round-trip. This is a consequence of staying close to the existing key schedule for KEMTLS.

Allowing `ServerFinished` to be transmitted immediately after `ServerKemCiphertext` and deriving SATS then would allow the server to initiate transmitting data sooner. This would reduce the overhead to an extra half round-trip, but rely on implicit authentication. This change however greatly complicates the key schedule, as `ServerFinished` would no longer be sent last.

We might also allow the client to send `ClientFinished` immediately after `ClientCertificate`. The client would then derive CATS without mixing in ss_C. This would not introduce extra round-trips before the client can send data, but the data that the client sent can then not be straightforwardly authenticated.

E XMSS AT NIST SECURITY LEVEL 1

The security of XMSS parameter sets specified in [44] reach NIST security level 5 (equivalent to AES-256) and above. This high level of security has only a very minor impact on computational performance, but it does have a significant impact on signature size. The draft of the NIST standard also considers parameter sets targeting security level 3 (equivalent to AES-192); the simple modification is to truncate all hashes to 192 bits. The extension to a parameter set targeting NIST level 1 is straight-forward: hashes are simply truncated to 128 bits; we obtain this by using SHAKE-128 [67] with 128 bits of output.

We define $\text{XMSS}^{\text{MT}}_{\text{s}}$ as an instantiation of XMSS^{MT} using two trees of height 12 each, i.e., a total tree height of 24, which limits

the maximum number of signatures per public key to $2^{24} \approx 16.7$ M. Increasing this maximum number of signatures to, for example, $2^{30} \approx 1$ billion increases signature size by only 96 bytes and has negligible impact on verification speed. It does have an impact on key-generation speed and signing latency, but as mentioned in Section 6.4, latency of signing is not very relevant when used by certificate authorities as in our paper.

Multi-tree XMSS is particularly well-suited for efficient batch signing. The idea is to compute one whole tree (of height h/d) on the lowest level and use it on-the-fly to sign $2^{h/d}$ messages. The computational effort per signature is then essentially reduced to one WOTS$^+$ key-pair generation.

We set the Winternitz parameter in $\text{XMSS}_\text{s}^\text{MT}$ to $w = 256$ to optimize for signature size. Changing to the more common $w = 16$ would increase signature size by about a factor of 2 and speed up verification by about a factor of 8.