A tale of two models: Formal verification of KEMTLS via Tamarin

(Full version)

Sofía Celi
Brave Software[*]
cherenkov@riseup.net

Jonathan Hoyland
Cloudflare, Inc.
jhoyland@cloudflare.com

Douglas Stebila
University of Waterloo, Ontario, Canada
dstebila@uwaterloo.ca

Thom Wiggers
Radboud University, Nijmegen, The Netherlands
thom@thomwiggers.nl

26th August 2022

KEMTLS is a proposal for changing the TLS handshake to authenticate the handshake using long-term key encapsulation mechanism keys instead of signatures, motivated by trade-offs in the characteristics of post-quantum algorithms. Prior proofs of security of KEMTLS and its variant KEMTLS-PDK have been hand-written proofs in the reductionist model under computational assumptions. In this paper, we present computer-verified symbolic analyses of KEMTLS and KEMTLS-PDK using two distinct Tamarin models. In the first analysis, we adapt the detailed Tamarin model of TLS 1.3 by Cremers et al. (ACM CCS 2017), which closely follows the wire-format of the protocol specification, to KEMTLS(-PDK). We show that KEMTLS(-PDK) has equivalent security properties to the main handshake of TLS 1.3 proven in this model. We were able to fully automate this Tamarin proof, compared with the previous TLS 1.3 Tamarin model, which required a big manual proving effort; we also uncovered some inconsistencies in the previous model. In the second analysis, we present a novel Tamarin model of KEMTLS(-PDK), which closely follows the multi-stage key exchange security model from prior pen-and-paper proofs of KEMTLS(-PDK). The second approach is further away from the wire-format of the protocol specification but captures more subtleties in security definitions, like deniability and different levels of forward secrecy; it also identifies some flaws in the security claims from the pen-and-paper proofs. Our positive security results increase the confidence in the design

[*]This work was done while the author was employed at Cloudflare, Inc.

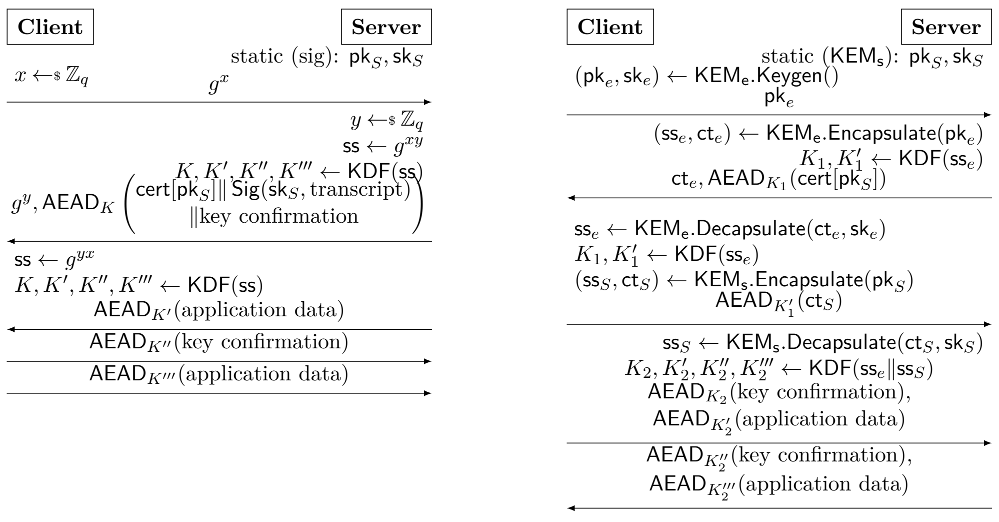

Figure 1: Simplified protocol diagrams of server-only authenticated versions of: (left) the TLS 1.3 handshake, using signatures for authentication; and (right) the KEMTLS handshake, using KEMs for authentication.

of KEMTLS(-PDK). Moreover, viewing these models side-by-side allows us to comment on the trade-off in symbolic analysis between detail in protocol specification and granularity of security properties.

1. Introduction

The Transport Layer Security (TLS) protocol is one of the most used cryptographic protocols. In its most recent version, the TLS 1.3 [32] handshake employs an ephemeral (elliptic-curve) Diffie–Hellman (DH) key exchange to establish session keys for confidentiality. In the regular handshake, TLS 1.3 authenticates the server and optionally the client using RSA or elliptic-curve signatures. It transmits the public keys to verify those signatures during the handshake, in certificates signed by a certificate authority (CA).

KEMTLS [33] is an alternative proposal for a post-quantum TLS 1.3 [32] handshake. It avoids using handshake signatures, which typically authenticate the TLS 1.3 handshake, replacing them with end-entity authentication based on key encapsulation mechanisms (KEMs) following well-established techniques for implicitly authenticated key exchange. As post-quantum KEMs are typically more efficient than the post-quantum signature schemes, either in bytes on the wire or computational efficiency, this saves resources. KEMTLS-PDK ("pre-distributed public key") [35] is a variant of KEMTLS that offers a more efficient handshake if the client already has the server's long-term public key. The authentication mechanisms from KEMTLS and KEMTLS-PDK have been proposed for standardisation to the Internet Engineering Task Force (IETF) TLS working group [13].

Figure 1 shows the cryptographic core of the unilaterally authenticated TLS 1.3 and KEMTLS handshakes. KEMTLS replaces the TLS 1.3 Diffie–Hellman-based ephemeral key exchange by KEM operations. Most importantly, whereas in TLS 1.3 the server authenticates by signing the transcript using the key from the server's certificate, in KEMTLS the client encapsulates against the KEM public key in the server's certificate. KEMTLS then combines both KEM shared secrets—one from the ephemeral key exchange and one from the server's long-term key—to derive a key that is *implicitly authenticated*, meaning only the intended server will be able to derive the secret. The client can then use the derived key to transmit application data.

At many levels, the KEMTLS handshake is similar to the TLS 1.3 handshake. However, due to the usage of KEMs, the order of messages in TLS 1.3 has been significantly changed. Additionally, the server can no longer send data in its first response to the client. However, KEMTLS preserves the client's ability to send its message after receiving the first flight from the server.

As KEMTLS is a novel way to achieve authentication in the TLS 1.3 handshake, the security of its design should be carefully checked not only with pen-and-paper proofs but with a computer-assisted formal analysis of it to provide stronger evidence of its soundness to adopters and standarization bodies like the IETF.

1.1. Related work

Analysis of TLS 1.3. During the development process of TLS 1.3, there was a strong collaboration between the standardisation community with the academic research community. Initial TLS 1.3 protocol designs were based on academic designs [29], and it was explicit goal of the TLS 1.3 process to incorporate academic security analysis of new designs before continuing with standarisation. Paterson and van der Merwe described this as a "design-break-fix-release" process rather than the "design-release-break-patch" cycle that was found on prior versions of the standarisation and usage of TLS [31]. Many of the security analyses of TLS 1.3 used the reductionist security paradigm [19–21, 28, 29]. Complementing this manual proof work, computer-aided cryptography [1] was also instrumental in checking TLS 1.3. Analyses were done using the Proverif [7] and Tamarin [14,15] symbolic analysis tools, as well as a verified implementation in F* [16].

Analysis of KEMTLS. The initial KEMTLS and KEMTLS-PDK papers included reductionist security proofs [33, 35], adapting the multi-stage key exchange approach used by Dowling et al. [19, 20] for TLS 1.3. Subsequently, Towa et al. proposed and proved an alternative abbreviated handshake, with additional short-lived static keys [25], and found a few minor mistakes in the original security proofs, which were subsequently fixed in online versions of the original papers [34, 36]. All these proofs treat protocol modes independently—one-at-a-time—and do not consider the presence of the other protocol modes.

1.2. Contributions

In this work, we present two security analyses of all four variants of KEMTLS (the base KEMTLS protocol, with server-only or mutual authentication, and the pre-distributed public keys variant KEMTLS-PDK, also with server-only or mutual authentication) using Tamarin [3,30]. The source code of our models is available at https://github.com/kemtls/.

Our first model, presented in Section 3, is based on the Tamarin analysis of TLS 1.3 by Cremers et al. [14]. This is a highly detailed model in terms of the protocol specification, closely following the TLS 1.3 wire format. In this model, we show that all four KEMTLS variants have equivalent security properties to the main handshake of TLS 1.3 without extensions. In implementing this model for KEMTLS, we were able to fully automate the proof, unlike the original model which required significant manual effort.

Our second model, presented in Section 4, is a novel Tamarin model developed from scratch that closely follows the multi-stage key exchange security model used in the pen-and-paper proofs [33,35]. This model focuses on the "cryptographic core", meaning that it is further away from the wire specification and does not model details like message encryption or the record layer. However, it captures more details in the security definitions, using the more granular definitions of forward secrecy from [33,35] as well as including an analysis of deniability. This model allows us to symbolically verify the reductionist security claims from the pen-and-paper proofs, but goes further by considering all four KEMTLS variants simultaneously. This Tamarin model allowed us to identify some minor flaws in the properties stated based on pen-and-paper proofs.

In Section 5, we compare the features of our two Tamarin models. Having these two models side-by-side illustrates the trade-off between detail of protocol specification and granularity of security properties. Ideally, of course, one would achieve both levels of detail simultaneously, but such complexity is challenging both for the humans reading and writing pen-and-paper proofs or authoring Tamarin models, and for computers checking such Tamarin models (where runtime typically scales exponentially with the complexity of the model). Our side-by-side approach with two very different perspectives still yields significant confidence in the soundness of the KEMTLS protocol design and each provides insight into flaws in the earlier models that it was based on.

2. Background on symbolic analysis

One approach to proving the security properties of protocols is *symbolic analysis*, which uses formal logic to reason about the properties of an algebraic model of a protocol. Computational tools, such as Tamarin [3,30] or ProVerif [9], can then be used to check whether certain properties hold in the symbolic model.

In symbolic analysis, generic symbols replace specific values. Operations like encryption are also modelled symbolically: for example, `senc(a,b)` represents the value `a` being symmetrically encrypted with the key `b`. In symbolic analysis, cryptographic operations are *perfect*, meaning the adversary can learn nothing about an encrypted message without the correct key. The operations that describe a protocol in a symbolic model take messages and state information, and transform them into the next state or emit another protocol message. A tool can then use all operations and symbols to generate every possible protocol run.

Many symbolic analyses of protocols use the Dolev–Yao [18] attacker model, in which an attacker can manipulate all messages at will, e.g. by redirecting them, replaying them, dropping them, or manipulating their contents. It can also construct new messages from information previously learnt. However, as the cryptography is assumed to be perfect, the attacker can not read or modify encrypted or authenticated messages if it does not have the right keys.

Symbolic models can also be extended to give the attacker special extra abilities. For example, one can allow the attacker to reveal private keys or state information of parties by performing queries to a reveal oracle. We record when the attacker uses this oracle, so reveal queries become part of the trace of execution.

Security properties are modelled as predicates over execution traces. In Tamarin, during the execution of the rules of the protocol, we can emit *action facts*. We use these action facts to record, for example, the session's impression of the authentication status or the current keys. We then write lemmas representing security properties as predicates over action facts: for example, that any key recorded in a certain type of action fact must not be known to the adversary, unless the adversary cheated by revealing keys. A model checker like Tamarin can then be used to check if the protocol maintains the required security property. Assuming soundness of the tool, either the tool will give a proof that the protocol has the required property, find a counter-example, or fail to terminate.

3. Model #1: high-resolution protocol specification

In this section, we discuss the natural approach of taking one of the TLS 1.3 models and adapting it to KEMTLS(-PDK). Our work demonstrates that KEMTLS provides security guarantees at least equivalent to those proven by Cremers et al. for the main handshake of TLS 1.3.

3.1. Cremers et al.'s Tamarin TLS 1.3 model

The Tamarin model of TLS 1.3 [14] is very high-resolution in terms of its modelling of protocol details and adherence to the protocol specification. It covers the cryptographic computations such as the key exchange and the key schedule; for example, calls to HKDF are decomposed into hash function calls. This model also includes the extensions to the basic TLS 1.3 handshake, such as the `HelloRetryRequest` mechanism, pre-shared keys, and resumption via session tickets. Additionally, it models the encryption of handshake messages, the syntax of the protocol messages, and mechanics such as TLS 1.3 extensions.

In terms of security properties, the Cremers et al. model extends Tamarin's basic Dolev–Yao attacker with the ability to recover secrets from Diffie–Hellman key shares and reveal the long-term keys of participants. TLS 1.3 is not secure against an attacker who can use these attacks freely, but aims to provide confidentiality and integrity against an attacker who is restricted from revealing secrets of the target session. Cremers et al. were able to encode lemmas capturing most of the security properties claimed by the TLS 1.3 specification [32, Appendix E.1]. They report that proving all lemmas in their model took about a week. Much of this time was spent on manual interaction with Tamarin's prover to guide it to prove some of the more complex lemmas. Verifying the generated proof requires "about a day" and "a vast amount of RAM" [14].

3.2. Representing KEMTLS in the model

We now describe how we modified the existing TLS 1.3 model to represent both KEMTLS and its variant with pre-distributed keys, KEMTLS-PDK. The original model is highly modular, which made it relatively easy to modify.

3.2.1. Modeling KEMs

Tamarin does not have a built-in interface to model KEMs. They can be described using Tamarin's asymmetric encryption primitives, as was done in [26]. We choose to model the KEM interface using Tamarin's function API. As the TLS 1.3 model has some support for cryptographic agility in the ephemeral key exchange, we add a public algorithm identifier symbol to each operation. We use fresh values to resemble KEM secret keys, and a function `kempk/2` to represent the public key for the specified algorithm. Tamarin's functions are just symbols and do not describe functionality. Any functionality is handled by writing equations over the symbols. As such, we can not return two values or generate fresh values in the encapsulate operation. We resolve this by providing a shared secret as an input to the encapsulate operation. The shared secret is defined through `kemss/2` and also contains the algorithm symbol. We define `kemencaps/3` over the algorithm, the shared secret and the public key. The resulting ciphertext can be provided to `kemdecaps/3` with the algorithm and the secret key. We model the functionality of `kemencaps` and `kemdecaps` by the following equation, where `alg` represents the KEM algorithm, and `seed` and `sk` are the fresh input values:

```
equations:
  kemdecaps(alg,
            kemencaps(alg,
                      kemss(alg, seed),
                      kempk(alg, sk)),
                      sk)
  = kemss(alg, seed)
```

3.2.2. Modelling KEMTLS

The model of Cremers et al. represents TLS 1.3 through rules that manipulate a specific state object, which keeps of many protocol variables, such as keys, authentication status, and the currently active handshake mode. Tamarin rules create transitions between these states. Where the protocol branches, such as when the server requests client authentication by sending `CertificateRequest`, there are two rules that end up in the same next state; for example, they would set the `cert_req` variable differently. The server later uses this variable to decide which of the rules `recv_client_auth` or `recv_client_auth_cert` to use; the latter expects the `Certificate`, `CertificateVerify`, and `Finished` messages, while the former only expects `Finished`. We handle the public key infrastructure (PKI) for KEM public keys in the same way as [14]: we do not model CA certificates, and assume an out-of-band binding between public keys and identities.

Ephemeral key exchange in the TLS 1.3 model uses Tamarin's Diffie–Hellman functionality. It also allows the negotiation of two different DH groups. During the handshake, the client and server generate ephemeral DH secrets for the chosen group. If the server rejects the client's choice of DH group, it falls back to another group through the `HelloRetryRequest` mechanism. To model the post-quantum ephemeral key exchange in KEMTLS, we replaced the Diffie–Hellman operations by `kemencaps` (KEM encapsulation) in place of the server's DH key generation. The client then computes the shared secret via `kemdecaps` (KEM decapsulation).

The authentication rules and states required more careful consideration. In the TLS 1.3 model, the `Certificate`, `CertificateVerify`, and `Finished` messages were sent and received simultaneously. In KEMTLS, we split the handling of these messages, as the peer that is authenticating needs to first receive a ciphertext to decapsulate. Doing this requires more states. Additionally, in KEMTLS the client sends `Finished` before the server, which deviates from TLS 1.3.

To finish our integration of KEMTLS, we made changes to the key schedule to include the computation of KEMTLS' Authenticated Handshake Secret (AHS) and use the correct handshake traffic encryption keys. We also modified the action facts emitted in the various rules to match our KEM operations; lemmas that made use of these action facts were also updated. We disabled the PSK and session ticket features of the original model.

3.2.3. Modeling KEMTLS-PDK

In KEMTLS-PDK, the client has the server's long-term public key beforehand. Access to the public key allows the client to send a ciphertext in the initial `ClientHello` message. Additionally, the client may attempt client authentication proactively and thus transmit its `Certificate` before receiving `ServerHello` from the server. We model this through an additional initial state for the KEMTLS-PDK client. From this state, there are two rules which set the state variable that will decide if the client will send its certificate. KEMTLS-PDK is otherwise implemented as a mostly separate sequence of states and rules, as the key schedule and order of messages are quite different. The client and server still transition through a state shared with KEMTLS, so they can fall back to the "full" handshake.

3.3. Security properties

We adapt the lemmas from the Cremers et al. model for TLS 1.3. Many core lemmas are constructed around the `SessionKey` fact: the client and the server record this fact when the handshake concludes. `SessionKey` contains the actor's final understanding of its and its peer's identities, authentication statuses, and the application traffic keys. We prove

```
lemma secret_session_keys:
  "All tid actor peer kw kr pas #i.
    SessionKey(tid, actor, peer, <pas, 'auth'>, <kw, kr>)@#i &
    not(Ex #r.RevLtk(peer)@#r & #r < #i) &
    not(Ex tid3 esk #r.RevEKemSk(tid3, peer, esk)@#r & #r < #i) &
    not(Ex tid4 esk #r.RevEKemSk(tid4, actor, esk)@#r & #r < #i)
    ==> not Ex #j. K(kr)@#j"
```

Listing 1: The `secret_session_keys` lemma proves application traffic keys are secret.

all security properties discussed in [14], and briefly explain the most important of these below.

3.3.1. Adversary compromise of secrets

First, we note the extent to which the adversary can compromise ephemeral or long-term secrets. KEMTLS uses ephemeral KEM keys for ephemeral secrecy and long-term KEM keys for authentication. The adversary can reveal actors' long-term secret keys; this records the `RevLtk($actor)` fact. We also allow revealing the ephemeral secret key in individual sessions, recording the `RevEKemSk(tid, $actor, esk)` fact. Variables `tid` ("thread identifier") and `esk` track the specific session and secret key.

KEMs are not "symmetric" in the same way that Diffie–Hellman key exchange is. Only one party in each KEM key exchange has a secret key that can be targeted by a reveal query. We do not model revealing the shared secret from the ciphertext.

Intermediate session keys, like the Main Secret (MS), can not be revealed directly. This follows from the design of the original model: in TLS 1.3, these secrets only depend on the ephemeral key exchange, so revealing the ephemeral key exchange in sessions not targeted by a lemma still allows the adversary to obtain those sessions' intermediate session keys. In KEMTLS, this is no longer the case: we mix the shared secrets encapsulated against long-term keys into the key schedule; as a result, our attacker is slightly weaker. The model discussed in Section 4 does directly allow session key reveal.

3.3.2. (Forward) secrecy of session keys

The outputs of the handshake, as recorded in the `SessionKey` fact, are the application traffic read and write keys `kr` and `kw`. We require these keys to remain secret against various forms of attacks. Forward secrecy requires that if the long-term keys (but not the ephemeral keys) were compromised after the session completes, the session keys remain secure.

We model this in the `secret_session_keys` lemma as shown in Listing 1. This lemma considers a client or server that believes it has authenticated its peer, where the attacker has not revealed the ephemeral KEM secret keys. We allow the attacker to reveal the peer's long-term secret key, but only after the `SessionKey` fact was emitted; this is the "forward" secrecy aspect. The attacker should not be able to learn (`not Ex #j. K(kr)@#j`) the

```
lemma secret_session_keys_ephem_reveal:
  "All tid actor peer kw kr pas #i.
    SessionKey(tid, actor, peer, <pas, 'auth'>, <kw, kr>)@#i &
    not (Ex #r. RevLtk(peer)@#r) &
    ==> not Ex #j. K(kr)@#j"
```

Listing 2: The `secret_session_keys_ephem_reveal` lemma proves application traffic keys are secret even if ephemeral keys are revealed.

target's read key `kr` under these constraints. We similarly prove forward secrecy for each of the intermediate keys in the key schedule: the Handshake Secret (HS), AHS and MS.

Note that in KEMTLS, the session keys are derived from not just the ephemeral key exchange as in TLS 1.3, but also include the secret encapsulated during the authentication phase of the handshake. This implies that both the ephemeral key and the server's long-term key need to be compromised in client sessions, and the ephemeral key and the server's long-term key in server sessions with mutual authentication. We prove this in our model through a variant of the `secret_session_keys` lemma that allows ephemeral key compromise, as long as the peer's long-term key is never revealed. This lemma is shown in Listing 2.

3.3.3. Authentication

We model the authentication properties of KEMTLS in the same way as they were modelled for TLS 1.3. The client and server are partnered via the nonces exchanged in the initial messages. The `entity_authentication` lemma captures that if the client, at the end of the handshake protocol, has authenticated their peer, and the peer's long-term keys have not been revealed, then there must be a peer session that started with the same nonces. This lemma is shown in Listing 3. The lemma `mutual_entity_authentication` states the same, but with the roles of client and server reversed. As these lemmas allow revealing the targeted actor's long-term keys, these properties also cover key-compromise impersonation attacks. Similarly, in the lemma `transcript_agreement`, we prove that when the client, after receiving the server's `Finished` message, commits to a transcript, there exists a server that is running with the same transcript (or their long-term keys have been revealed). The `mutual_transcript_agreement` lemma states the same but with the roles reversed.

In TLS 1.3, the verification of the handshake signature immediately ensures authentication. In KEMTLS authentication is only made explicit when the `Finished` messages are verified. However, the lemmas in the original model already only captured authentication through the `Finished` messages.

3.4. Results

After adding relevant helper lemmas, Tamarin was able to auto-prove all the correctness and security lemmas for Model #1, with all four KEMTLS variants supported

```
lemma entity_authentication [use_induction]:
  "All tid actor peer nonces cas #i.
    commit(Nonces, actor, 'client', nonces)@i &
    commit(Identity, actor, 'client', peer, <cas, 'auth'>)@i &
    not (Ex #r. RevLtk(peer)@r & #r < #i)
    ==> (
      Ex tid2 #j_ea.
        running2(Nonces, peer, 'server', nonces)@j_ea
        & #j_ea < #i
    )"
```

Listing 3: The `entity_authentication` lemma proves that if a client commits to a set of nonces, there is a server that's running with the same nonces.

simultaneously. Run-times are shown in Appendix C.

3.4.1. Auto-proving and helper lemmas

Many of the lemmas in Cremers et al.'s model of TLS 1.3 were not able to be auto-proved by Tamarin; instead, the authors had to manually guide Tamarin through parts of the proof. Our goal was to improve the model so that it could be proved automatically, with no manual intervention required.

To help the automated prover, Cremers et al. introduced many intermediate lemmas, many of which state properties of earlier keys or more limited message exchanges. Inheriting these lemmas proved to be both helpful as well as distracting. Incrementally proving and adjusting the intermediate lemmas to apply to KEMTLS(-PDK) helped us spot bugs and make progress. But starting from their helper lemmas often left us unclear as to why particular intermediate lemmas were necessary to prove the final security properties.

In our experience, Tamarin does not find counterexamples very easily in big models. As a result, we wrote increasingly "smaller" lemmas whenever we ran into a lemma that was hard to prove. This greatly expanded the number of helper lemmas available. While we believe that this helped auto-prove the model, it also resulted in cases where the helper lemmas interacted in bad ways and had to be ignored. (Replacing Diffie–Hellman by KEM, thus avoiding Tamarin's algebraic analysis of DH group operations, may also have eased analysis.) Additionally, the model of [14] is carefully split over different files to avoid certain helper lemmas from interacting. With much less experience, we joined together most of those files, which in many cases lead to Tamarin getting distracted by helper lemmas. Many hours in the manual prover helped us determine which lemmas needed to be marked by `ignore_lemma` annotations. While doing this, it was often helpful to rename lemma variables to be distinguishable, as Tamarin does not indicate what lemmas it tries to apply. For example, to identify the lemma `entity_authentication`, we renamed a time variable `#j` to `#j_ea`.

3.4.2. A bug in Cremers et al.'s TLS 1.3 lemmas

While working on the proof, we found that one of the core lemmas in [14]'s TLS 1.3 model seems to have changed after creating the proof. The lemma `session_key_agreement` tried to prove that the client's and servers values of `keys` in the `SessionKey` fact matched. However, variable `keys` is a tuple `<kr, kw>` of the reading and writing keys of each peer. As the server's writing key should match the client's reading key and not the client's writing key, this lemma did not hold. The rendered proofs included in the repository alongside the model and lemmas revealed that in the executed proof, `keys` was split into its elements and equated correctly. We disclosed the bug to the authors.

It is not hard to imagine how such a mistake slips into a model if re-proving the smallest changes requires days of manual proving effort. We view this as evidence of the value of auto-proving models: being able to let the computer "do its thing" allows us to make changes more confidently.

3.5. Limitations

Although the model is very granular in its description of KEMTLS(-PDK), we do have some limitations. As discussed in Section 3.3.1, we do not model intermediate session key reveal. We also have not modelled session resumption or pre-shared key modes with KEMTLS. Finally, we have not attempted to model deniability, which we will model in Section 4.

4. Model #2: multi-stage key exchange model

The security properties shown in the original KEMTLS paper [33] and the KEMTLS-PDK paper [35] are stated using the reductionist security paradigm, via the *multi-stage key exchange model* [24], which was adapted for proofs of the TLS 1.3 handshake [19, 20]. Our goal in this section is to translate the reductionist security properties in this model—match security, session key indistinguishability, and authentication—from a pen-and-paper model to being encoded in Tamarin, then have the Tamarin prover confirm these properties hold. Notably, this model discriminates between the several keys established within a single KEMTLS handshake, associating distinct security properties with individual stage keys.

4.1. Reductionist security model for TLS 1.3 and KEMTLS

The multi-stage key exchange security model, first introduced by Fischlin and Günther [24], is an extension of the Bellare–Rogaway (BR) model [6] for proving security of authenticated key exchange in the reductionist security paradigm. In the BR model, the adversary is in control of all communications between honest parties, so the adversary can activate honest parties to send their next protocol message, and can also modify, delay, drop, replay, or create messages. Each honest party can run multiple simultaneous or sequential executions of the protocol (each execution at a party is called a session) sharing a single long-term key pair across their sessions. Within each session, a party maintains several variables, including the execution status, a session identifier, an identifier for the peer (if the peer is to be authenticated), and a session key. The adversary interacts with the honest parties via oracles, including oracles for starting a new session at a party (the NewSession oracle) and message delivery and response (Send), as well as letting the adversary learn an honest party's long-term key (Corrupt) or the session key of a particular session (Reveal). The adversary may choose one session as a challenge session and, via a call to the Test oracle, receive an indistinguishability game challenge. This challenge is either the real session key established in that session or a value, chosen uniformly at random, from the space of all session keys. The adversary's task is to distinguish whether it was given a real or random value in the Test query, under the condition that the tested session remains *fresh*, meaning certain combinations of reveal queries were not used. The freshness condition can be tweaked to capture different characteristics such as forward secrecy. The BR model can also include an explicit authentication property, which checks whether the adversary can cause one party to accept a session as authenticated with a particular intended peer, without, under some conditions, that peer having participated in a corresponding session.

There are many extensions to the BR model to capture different functionality and security properties; see [10, Ch. 2] for a summary. One important extension is the formalisation by Brzuska et al. [11,12] which introduces a property called *match security*. This checks the technical condition that the session identifiers specified by the protocol effectively match the partnered sessions. Among other benefits, match security helps with composition theorems involving AKE protocols.

In real-world protocols like QUIC and TLS, as well as KEMTLS, multiple keys are established in each session for different purposes. Fischlin and Günther [24] created the multi-stage key exchange model, an extension of the BR model in which a single session can have multiple stages, each of which establishes a key with certain security characteristics; they used this approach to analyse QUIC. It was also used by Dowling et al. [19, 20] to analyse the TLS 1.3 handshake. As KEMTLS is an alternative realization of the TLS 1.3 handshake, it is natural to similarly use this model for analyzing KEMTLS, as done in [33].

We now present the technical components of the multi-stage key exchange security model as used in KEMTLS [33] and KEMTLS-PDK [35]. Our presentation here will be somewhat abbreviated; for full details, see the full versions of [33, 35].

4.1.1. Partnering

For the proof of KEMTLS(-PDK), we need to keep track of the pairs of sessions that are (supposedly) communicating. Each session keeps track of per-stage session identifiers, each of which is a distinct label for the stage followed by all plaintext messages transmitted up until that point in the protocol; for KEMTLS-PDK, this also includes the implicit `ServerCertificate` message. We call two sessions *partners* if their session identifiers match.

4.1.2. Adversary interaction

The oracles and variables stated in Section 4.1 suffice for modelling the various properties of match-security. To model key indistinguishability, the multi-stage model includes an oracle $\mathsf{Test}(\pi, i)$ which challenges the adversary to distinguish the ith stage key of session π from random.

4.1.3. Match security

There are six specific properties in the multi-stage match security definition used for KEMTLS [33]. For distinct sessions π, π' which are partnered in some stage i:

1. π and π' agree on the same key at every stage $j \leq i$;
2. π and π' have opposite roles;
3. π and π' are also contributive partners in stage i;
4. for stages $j \leq i$ of π which are considered explicitly authenticated if stage i has accepted, the identity of π's peer is correct;
5. in any two (not necessarily distinct) sessions, distinct stages have distinct session identifiers;
6. there is not a third distinct session that is also partnered to π and π' in stage i.

Match security properties are often proved information-theoretically rather than under cryptographic assumptions, as the structure of a well-designed protocol and its instantiation within the formalism usually quite naturally yields the desired match security properties.

4.1.4. Multi-stage security and malicious acceptance

Multi-stage security models secrecy of each stage key under specific forward secrecy properties. These properties include *implicit* and *explicit authentication.* The model is parameterized by values indicating the expected security properties of particular stage keys. [33, 35] define four levels of forward secrecy:

- No forward secrecy (0);
- *Weak forward secrecy level 1* ($\mathsf{wfs1}$): the key is confidential against passive adversaries. This level allows the adversary to access the peer's long-term keys. Keys with this level of forward secrecy have no authentication.
- *Weak forwards secrecy level 2* ($\mathsf{wfs2}$): the key is confidential against passive adversaries ($\mathsf{wfs1}$) and against active adversaries who never corrupted the peer's long-term key. In the latter case, the key is implicitly authenticated.

- *Forward secrecy* (fs): the key is confidential against passive adversaries ($\mathsf{wfs1}$) and against active adversaries who did not corrupt the peer's long-term key before the stage accepted. Keys with forward secrecy level fs are implicitly authenticated.

As the protocol is executed, the security level of a particular stage key may be upgraded once a later stage acceptsthe server's security levels are different if mutual authentication is used.

In Eq. (1), $\mathsf{FS}_{i,j}$ gives the expected forward secrecy security of a KEMTLS client session's stage i key, assuming that stage j has accepted.

$$\mathsf{FS} = \begin{pmatrix} \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} \\ & & & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} \\ & & & & \mathsf{wfs2} & \mathsf{fs} \\ & & & & & \mathsf{fs} \end{pmatrix} \tag{1}$$

Explicit authentication, which is e.g. achieved by the `Finished` messages, is modelled through *malicious acceptance*: an adversary should not be able to cause a supposedly explicitly authenticated stage to accept without a partner stage.

4.1.5. Deniability

Roughly speaking, deniability is the property that a party cannot provide proof to a judge that a peer participated in a particular protocol execution, even if they did. First introduced in general by Dwork, Naor, and Sahai [23] and in the context of key exchange by Di Raimondo, Gennaro, and Krawczyk [17], there are many flavours and variations of deniability; see e.g. [27] for a classification. *Offline deniability* is the inability of a judge to distinguish between a transcript generated by honest parties and a transcript generated by a simulator. The form of deniability offered by KEMTLS and KEMTLS-PDK (following the terminology of [27]) is that it provides offline deniability in the *universal deniability* setting (meaning the simulator only has access to parties' long-term public keys) against an unbounded judge with full corruption powers (meaning the judge gets the parties' long-term secret keys as well as any per-session coins).

4.1.6. Pen-and-paper proofs

The KEMTLS and KEMTLS-PDK papers [33, 35] provide theorems and give proofs that their respective protocols satisfy the match-security and multi-stage security properties; they do not include any proofs for offline deniability. The match-security properties are shown information-theoretically, with terms depending on the number of sessions, the correctness probability of the KEMs, and the size of the TLS nonce space. The multi-stage security properties are shown under the following computational assumptions: hash function collision resistance, IND-1CCA security of $\mathsf{KEM_e}$, PRF and dual-PRF security of HKDF.Extract, PRF-security of HKDF.Expand, EUF-CMA security of HMAC, and IND-CCA security of $\mathsf{KEM_c}$ and $\mathsf{KEM_s}$. There is a tightness loss proportional to the number of sessions squared.

4.2. Formalizing the reductionist security model in Tamarin

We formalized all four KEMTLS variants (regular and PDK, server-only and mutually authenticated) in Tamarin, along with lemmas capturing correctness, match security, multi-stage security, and deniability, analogous to the definitions from Section 4.1. We now describe the formalization in more detail.

4.2.1. Protocol description

This Tamarin formulation of the four KEMTLS variants focuses on the “cryptographic core” of the protocol. Roughly speaking, this is the protocol as formulated in figures in the original papers [33, 35], which includes cryptographic operations involved in the key exchange, but does not include extra fields and operations arising from the integration of the cryptographic operations into a network protocol. We only address the handshake protocol and exclude TLS message formatting, algorithm negotiation, and data structures such as certificates. We exclude extensions such as TLS 1.3 session resumption or pre-shared key handshakes. We assume that long-term public keys are reliably distributed out-of-band. We omit modelling handshake encryption: while the various handshake traffic secrets are established and recorded as accepted in each stage of the protocol, subsequent handshake messages are sent in plaintext. The various primitives based on hash functions ($\mathsf{HMAC}, \mathsf{HKDF.Extract}, \mathsf{HKDF.Expand}$) are modelled as independent opaque functions, rather than relying on each other and ultimately on a common hash function. As in the pen-and-paper proofs, there are three KEMs, $\mathsf{KEM_e}$, $\mathsf{KEM_c}$, and $\mathsf{KEM_s}$, for ephemeral key exchange, client authentication, and server authentication, respectively. The KEMs are modelled as distinct primitives, meaning that a party cannot use its long-term credential to act as both a client and a server.

4.2.2. Adversary interaction

Among the oracles stated in Section 4.1, the NewSession and Send oracles are not needed, since the Tamarin model includes rules for each protocol step. The Tamarin model does include Corrupt and Reveal oracles. Because key security in Tamarin is modelled not using indistinguishability but key recovery (the `K(...)` fact in Tamarin lemmas), there is no need for the Test query in the Tamarin model.

4.2.3. Definition of cryptography

For each KEM KEM_x, the following functions are defined in Tamarin: `KEM_x_PK/1` (to generate a public key from a secret key), `KEM_x_Encaps_ct/2` (to generate a ciphertext from a public key and random coins), `KEM_x_Encaps_ss/2` (to generate, during encapsulation, a shared secret from a public key and random coins), and `KEM_x_Decaps/2` (to decapsulate a ciphertext using a secret key to recover a shared secret). (Two `Encaps` functions are provided because functions in Tamarin only output a single value, so we need two functions to represent the two outputs from encapsulation.) Rewriting rules are provided to model that decapsulation with the appropriate values arrives at the same shared secret as encapsulation.

There are distinct functions for HKDF (`HKDFExtract/2` and `HKDFExpand/3`), HMAC (`HMAC/2`), and the hash function (`H/1`). We do not attempt to model the fact that HKDF is built from HMAC and that HMAC uses the same hash function `H`; they are all assumed to be independent. The HKDF API is simplified to not include a length parameter as input.

4.2.4. Correctness lemmas

We include a collection of "reachability" lemmas which check that, for every stage in all 4 protocol variants, it is possible to arrive at that stage, with honest client and server sessions having correct owner and peer information, matching contributive and session identifiers, and correct expectations on authentication, forward secrecy, and replayability; the reachability lemmas include checking retroactive upgrading of properties. These lemmas are implemented using Tamarin's `exist-trace` feature. There are 47 reachability lemmas in total, generated from a template using the M4 macro language.

We also include lemmas that check that the attacker works, in the sense that the attacker can successfully compute session keys of all stages by using the corruption oracles.

4.2.5. Security and authentication lemmas

The match security lemmas from Definition B.1 of [33], plus the adjustments for replayability in [35], are directly translated into Tamarin. The lemmas are basically predicates over the session-specific variables defined in the model syntax, and can be stated analogously since the Tamarin model includes action facts for each session-specific variable.

Session key security in Tamarin is modelled based on infeasibility of session key recovery, rather than indistinguishability of a session key from random. We have lemmas for each type of forward secrecy a stage key can have, directly translating the freshness conditions of [33, Defn. B.3] and [35, Defn. B.5].

We have a lemma for explicit authentication analogous to [35, Defn. B.5 3], including not requiring uniqueness of the replayable KEMTLS-PDK stage 1.

4.2.6. Deniability lemmas

Whereas the lemmas for the above properties all share the same Tamarin protocol description as explained above, the deniability lemmas use a re-statement of the protocol description. To formulate a deniability lemma, we need two versions of the protocol description: honest execution of the protocol using long-term secrets, and simulation using only public keys. The judge in the offline deniability game is passive and receives only transcripts, so we can collapse the multiple rules for each client and server action into a single rule that generates a full transcript including both client and server operations.

The deniability lemmas use Tamarin's observational equivalence feature [4] to check that the real and simulated transcripts are indistinguishable.

Deniability for each of the 4 KEMTLS protocol variants is dealt with separately. For each variant (e.g., KEMTLS with server-only authentication), we have one rule that generates real transcripts using long-term secret keys (e.g., `KEMTLS_SAUTH_real`) and one rule that generates simulated transcripts without long-term secret keys (e.g., `KEMTLS_SAUTH_simulated`). Finally, the adversary has access to a rule `real_vs_simulated` which takes as input one real transcript and one simulated transcript. It returns one of these to the adversary using Tamarin's `diff` operator for observational equivalence. By the running Tamarin prover with the `--diff` option to activate observational equivalence mode, Tamarin will check that it is not possible to distinguish which was given to the adversary.

Our definition of deniability is offline deniability in the universal deniability setting against an unbounded judge with full corruption powers. Consequently, the transcripts output includes the parties' long-term secret keys, the session keys computed in the real/simulated transcript, and the random coins allegedly used in the real/simulated transcript.

Using Taramin's observation equivalence feature causes a substantial increase in state space, so for efficiency reasons, we provide an option (using M4 macros) to omit portions of the transcript that are deterministically generated from earlier parts of the transcript and thus (from a mathematical perspective) could not help a distinguisher.

4.3. Comparison of pen-and-paper and Tamarin models

In principle, if the same security properties have been encoded in both a pen-and-paper reductionist security model and in a Tamarin model, a full and correct proof in the reductionist security model yields everything that a Tamarin proof could, and potentially more. In particular, reductionist security proofs do not idealize cryptographic primitives as much as Tamarin does. Moreover, a reductionist security proof can be done in the "concrete setting" [5], yielding a precise (non-asymptotic) relationship between the runtime and success probability of an adversary against the protocol versus the runtime and success probability of breaking the underlying cryptographic assumption. While it would be possible to encode the pen-and-paper proofs of KEMTLS from the original papers into a computer verification tool such as EasyCrypt [2], that would also require the cryptographer to manually write all game hops and reductions, a massive undertaking. To date, there are no proofs of KEMTLS using a computer-aided verification tool for reductionist proofs.

Tamarin does not lend itself to writing security properties in exactly the same way as would be used in reductionist security models. Although there is no way to objectively justify how close the pen-and-paper and Tamarin models of this section are to each other, subjectively we think they are quite close:

- The protocol specification in Tamarin maps nearly line-for-line onto the protocol figures in the original papers, using the same function interfaces, same key schedule, and same session identifiers.

- The session-specific variables in the pen-and-paper model correspond nearly one-for-one to action facts in the Tamarin model.
- There are Tamarin lemmas for each security property in the pen-and-paper model, and there is a clear mapping between the clauses in the predicates in the pen-and-paper model and the Tamarin model.

The main gap in modelling, as mentioned earlier, is that session key security is modelled via indistinguishability in the pen-and-paper models but via infeasibility of key recovery in the Tamarin model. Though it is possible to verify indistinguishability through Tamarin's observational equivalence features, the effect on the state space as discussed in Section 4.2.6 makes this impractical.

One other nice feature of our Tamarin models is that there is a fairly clean separation between protocol definition and security properties: files containing lemmas for match-security, multi-stage security, and authentication are phrased solely in terms of the action facts of the generic security model (similar to how a good pen-and-paper security model refers abstractly to the protocol API and model variables, rather than mixing in details of protocol instantiation), so these lemmas could be applied to any protocol in the same security model.

4.4. Results

Tamarin was able to auto-prove all the lemmas for correctness, reachability, match security, multi-stage session key security, authentication, and deniability in Model #2, with all four KEMTLS variants supported simultaneously. We did not need to create any helper lemmas for Tamarin. Run-times are shown in Appendix F.

4.4.1. Bugs in the original papers' security properties

When translating the models into Tamarin, we identified minor mistakes in some of the forward secrecy and authentication properties listed in the original KEMTLS [33] and KEMTLS-PDK [35] papers, highlighting the value of formal verification. We summarize the corrected properties in Appendix A.

4.5. Limitations

As noted above, the design of the model in this section imposes some limitations. Unlike in Section 3, we generally did not model non-cryptographic details of the handshake, such as TLS handshake messages, extensions, or the record layer. We also did not model handshake encryption or algorithm negotiation.

We also had, unlike in Section 3, three distinct KEMs for ephemeral key exchange, server authentication and client authentication. This implicitly assumes the same certificate is not used for both purposes, which was the basis of the Selfie attack [22]. Without this limitation, we observe a state-space explosion with a major impact on performance. For example, if $\mathsf{KEM}_c = \mathsf{KEM}_s$, the first 10 out of 11 `reachable_*` lemmas take over 8

Feature	**Model #1**	**Model #2**
Protocol modelling		
Encrypted handshake messages	✓	✗
HKDF and HMAC decomposed into hash calls	✓	✗
Key exchange and auth. KEMs are the same algorithm	✓	✗
TLS message structure	✓	✗
Algorithm negotiation	✓	✗
Security properties		
Adversary can reveal long-term keys	✓	✓
Adversary can reveal ephemeral keys	✓	✗
Adversary can reveal intermediate session keys	✗	✓
Secrecy of handshake and application traffic keys	✓	✓
Forward secrecy	✓	✓
Multiple flavours of forward secrecy	✗	✓
Explicit authentication	✓	✓
Deniability	✗	✓

Table 1: Comparison of features in our two Tamarin models of KEMTLS

hours, and the last `reachable_*` did not terminate after 45 hours, compared to all 11 `reachable_*` lemmas taking just over 1 minute with distinct KEM_c and KEM_s.

Our deniability lemmas are for abbreviated transcripts (without messages generated deterministically from earlier parts of transcript) and omit ephemeral coins. Again, without this limitation, there is a major impact on performance. For example, including full transcripts for KEMTLS-sauth increases run-time from 1 minute to 16 hours, whereas including ephemeral coins increases runtime from 1 minute to 110 minutes.

5. Comparison of models

We discussed two very different models of KEMTLS(-PDK) in the previous sections. These models are examples of how we can view modelling as the art of replacing specifics with generalities. Model #1 stays very close to the wire format of TLS 1.3 and phrases the security properties in terms of attacks on the ephemeral and long-term keys. It contains more implementation details such as algorithm negotiation, message framing, encryption of handshake messages, and even application data. Model #2 is more abstract in its representation of protocol messages. However, it models the cryptographic properties in a more granular fashion. This more abstract description closely follows the multi-stage pen-and-paper proofs of KEMTLS and KEMTLS-PDK, and allowed verifying the properties claimed in the pen-and-paper proofs. Table 1 summarizes differences between the two models, a few aspects of which we discuss further below.

5.1. Modelling KEMs

The two models differ in the way that they model the KEMs in the protocol. Model #1 uses the same functions for all KEM modes in the protocol (key exchange, server authentication and client authentication). Model #2 has three separate sets of functions for the three different KEM modes; this means the attacker can not copy ciphertexts or public keys from one of the modes to another, which should make proving the protocol easier. Interestingly, we saw significantly different performance between these two approaches. The second model proves in very short time with the three separate KEMs, but runtime blows up if we define all three KEM modes with the same functions; we did not attempt to generate the full proof because it took so long as discussed in Section 4.5. This suggests that splitting the three KEM modes in the first model could result in a speed-up. However, splitting the KEMs in Model #1 did not improve the time to auto-prove lemmas; in fact, a few lemmas even stopped being auto-provable. Ideally, this puzzle would be resolved with a justification that there is a way of safely separating uses of KEMs, allowing us to use whichever form happens to be easier for Tamarin to prove.

5.2. Threat model

Both models use Dolev–Yao attackers, but give the attackers slightly different extra abilities as noted in the bottom half of Table 1. Consequently, the results hold in slightly different circumstances. The attacker in Model #1 can compromise ephemeral keys and long-term keys, but not session keys, whereas the attacker in Model #2 can compromise intermediate session keys and long-term keys, but not ephemeral keys. Revealing the HS intermediate session key allows the second attacker to simulate the abilities of the first, but the reverse does not hold; the attacker in Model #2 is thus slightly stronger.

5.3. Ease of use

Work on each of our two models was done by separate authors of this paper, neither of whom had written a paper using Tamarin before and who had only had a basic introduction to Tamarin prior to this work. Surprisingly to us, creation of Model #2 from scratch was simpler and proceeded faster than the work in Model #1 adapting Cremers et al.'s TLS 1.3 model to KEMTLS. We attribute this to the higher fidelity of the protocol model in Cremers et al., requiring more code to model our changes, and the higher difficulty in proving.

6. Conclusion

We presented two Tamarin models checking security properties of KEMTLS and its variant protocol KEMTLS-PDK. Model #1 is highly detailed in implementation characteristics, close to the wire-format of the protocol. Model #2 presents the protocol at a higher level but provides a more precise characterization of security properties. We prove that

KEMTLS(-PDK) is secure in both models; importantly these analyses include all four KEMTLS variants supported simultaneously. Additionally, we proved offline deniability of KEMTLS(-PDK) in Model #2.

Overall, comparing these two analyses is something of an apples-to-oranges comparison. The two very different approaches allow us to model and test different properties of the protocol. Model #1 is closer to what an implementation would be like, and verifies the security properties in such a scenario. Adopting the Cremers et al. TLS 1.3 model [14] also allowed us to quickly adapt the security claims of TLS 1.3 to our protocols. Model #2, on the other hand, is an adaptation of the multi-stage authenticated key exchange model from the pen-and-paper proofs in [33, 35]. As such, Model #2 in a sense checks the claims in the pen-and-paper proofs, and in fact uncovered some minor mistakes in those proofs.

Our two models illustrate a common trade-off in formal analysis between the detail of the protocol specification and the granularity of the security properties we can prove. A similar observation was also made by Cremers et al. [14], who commented computational analyses could only look at parts of TLS 1.3, rather than considering all the modes at once.

While we proved certain privacy properties, such as deniability, our models can be further expanded to include other privacy properties, such as the proposed Encrypted Client Hello extension (previously called ESNI). These properties have only been proven by using the symbolic protocol analyzer ProVerif [8].

Acknowledgements

The authors gratefully acknowledge helpful suggestions from Peter Schwabe. D.S. was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grants RGPIN-2016-05146 and RGPIN-2022-03187. D.S. and T.W. were supported by an NLnet Assure grant for the project "Standardizing KEMTLS". T.W. was supported by the European Research Council through Starting Grant No. 805031 (EPOQUE).

References

[1] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-aided cryptography. In *2021 IEEE Symposium on Security and Privacy*, pages 777–795. IEEE Computer Society Press, May 2021. doi:10.1109/SP40001.2021.00008.

[2] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, Heidelberg, August 2011. doi:10.1007/978-3-642-22792-9_5.

[3] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, and Benedikt Schmidt. Tamarin prover, 2022. URL: https://tamarin-prover.github.io.

[4] David A. Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1144–1155. ACM Press, October 2015. doi:10.1145/2810103.2813662.

[5] Mihir Bellare. Practice-oriented provable-security. In *Proc. First International Workshop on Information Security (ISW 97)*, volume 1561 of *LNCS*, pages 221–231. Springer, 1998. doi:10.1007/BFb0030423.

[6] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994. doi:10.1007/3-540-48329-2_21.

[7] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy*, pages 483–502. IEEE Computer Society Press, May 2017. doi:10.1109/SP.2017.26.

[8] Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. Handshake privacy for TLS 1.3 - technical report. Research report, Inria Paris; Cloudflare, March 2022. URL: https://hal.inria.fr/hal-03594482.

[9] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96. IEEE Computer Society, 2001.

[10] Colin Boyd, Anish Mathuria, and Douglas Stebila. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, second edition, 2019. doi:10.1007/978-3-662-58146-9.

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

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

[13] Sofia Celi, Peter Schwabe, Douglas Stebila, Nick Sullivan, and Thom Wiggers. KEM-based Authentication for TLS 1.3. Internet-Draft draft-celi-wiggers-tls-authkem-01, Internet Engineering Task Force, September 2022. Work in Progress. URL: https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-01.

[14] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1773–1788. ACM Press, October / November 2017. doi:10.1145/3133956.3134063.

[15] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *2016 IEEE Symposium on Security and Privacy*, pages 470–485. IEEE Computer Society Press, May 2016. doi:10.1109/SP.2016.35.

[16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. Implementing and proving the TLS 1.3 record layer. In *2017 IEEE Symposium on Security and Privacy*, pages 463–482. IEEE Computer Society Press, May 2017. doi:10.1109/SP.2017.58.

[17] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Deniable authentication and key exchange. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 400–409. ACM Press, October / November 2006. doi:10.1145/1180405.1180454.

[18] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In *22nd FOCS*, pages 350–357. IEEE Computer Society Press, October 1981. doi:10.1109/SFCS.1981.32.

[19] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1197–1210. ACM Press, October 2015. doi:10.1145/2810103.2813653.

[20] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology*, 34(4):37, October 2021. doi:10.1007/s00145-021-09384-1.

[21] Benjamin Dowling and Douglas Stebila. Modelling ciphersuite and version negotiation in the TLS protocol. In Ernest Foo and Douglas Stebila, editors, *ACISP 15*, volume 9144 of *LNCS*, pages 270–288. Springer, Heidelberg, June / July 2015. doi:10.1007/978-3-319-19962-7_16.

[22] Nir Drucker and Shay Gueron. Selfie: reflections on TLS 1.3 with PSK. *Journal of Cryptology*, 34(3):27, July 2021. doi:10.1007/s00145-021-09387-y.

[23] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *30th ACM STOC*, pages 409–418. ACM Press, May 1998. doi:10.1145/276698.276853.

[24] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1193–1204. ACM Press, November 2014. doi:10.1145/2660267.2660308.

[25] Felix Günther, Simon Rastikian, Patrick Towa, and Thom Wiggers. KEMTLS with delayed forward identity protection in (almost) a single round trip. In *ACNS 2022*, 2022. to appear. URL: https://eprint.iacr.org/2021/725.

[26] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Philip R. Zimmermann. Post-quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy*, pages 304–321. IEEE Computer Society Press, May 2021. doi:10.1109/SP40001.2021.00030.

[27] Andreas Hülsing and Florian Weber. Epochal signatures for deniable group chats. In *2021 IEEE Symposium on Security and Privacy*, pages 1677–1695. IEEE Computer Society Press, May 2021. doi:10.1109/SP40001.2021.00058.

[28] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. (De-)constructing TLS 1.3. In Alex Biryukov and Vipul Goyal, editors, *INDOCRYPT 2015*, volume 9462 of *LNCS*, pages 85–102. Springer, Heidelberg, December 2015. doi:10.1007/978-3-319-26617-6_5.

[29] Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 81–96, 2016. doi:10.1109/EuroSP.2016.18.

[30] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013. doi:10.1007/978-3-642-39799-8_48.

[31] Kenny Paterson and Thyla van der Merwe. Reactive and proactive standardisation of TLS. In Lidong Chen, David A. McGrew, and Chris Mitchell, editors, *Security Standardisation Research (SSR) 2016*, volume 10074 of *LNCS*, pages 160–186. Springer, 2016. doi:10.1007/978-3-319-49100-4_7.

[32] Eric Rescorla. The Transport Layer Security TLS Protocol Version 1.3. RFC 8446, RFC Editor, August 2018. doi:10.17487/RFC8446.

[33] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1461–1480. ACM Press, November 2020. doi:10.1145/3372297.3423350.

[34] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. Cryptology ePrint Archive, Report 2020/534, 2020. https://eprint.iacr.org/2020/534.

[35] Peter Schwabe, Douglas Stebila, and Thom Wiggers. More efficient post-quantum KEMTLS with pre-distributed public keys. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021, Part I*, volume 12972 of *LNCS*, pages 3–22. Springer, Heidelberg, October 2021. doi:10.1007/978-3-030-88418-5_1.

[36] Peter Schwabe, Douglas Stebila, and Thom Wiggers. More efficient post-quantum KEMTLS with pre-distributed public keys. Cryptology ePrint Archive, Report 2021/779, 2021. https://eprint.iacr.org/2021/779.

A. Errors identified in the stated properties of KEMTLS(-PDK)

We identified minor mistakes in some of the forward secrecy and authentication properties listed in the original KEMTLS [33] and KEMTLS-PDK [35] papers. See the original papers for the definition of the symbols.

- In KEMTLS-mutual: $\mathsf{auth}_3^S = 3$ and $\mathsf{auth}_4^S = 4$ both should have been set to 5; $\mathsf{FS}_{3,3}^S = \mathsf{FS}_{3,4}^S = \mathsf{FS}_{4,4}^S = \mathsf{wfs2}$ should all have been $\mathsf{wfs1}$; and $\mathsf{auth}_6^S = 6$ should have been $\mathsf{auth}_6^S = \infty$.
- In KEMTLS-PDK-sauth: $\mathsf{FS}_{1,j}^C$ and $\mathsf{FS}_{1,j}^S$ should have been 0 for all j; $\mathsf{auth}_5^C = 5$ should have been $\mathsf{auth}_5^C = \infty$; and $\mathsf{FS}_{i,4}^S$ should have been $\mathsf{wfs1}$ for $i = 2, 3, 4$.
- In KEMTLS-PDK-mutual: the message `SKC` should have been included in the `SF` MAC computation and `SF` should have been included in the `CF` MAC computation; $\mathsf{FS}_{1,j}^C$ and $\mathsf{FS}_{1,j}^S$ should have been 0 for all j; $\mathsf{auth}_5^C = 5$ should have been $\mathsf{auth}_5^C = \infty$; and $\mathsf{FS}_{4,4}^S = \mathsf{wfs1}$ should have been $\mathsf{wfs2}$.

The online versions [34,36] of the source papers have been updated with our corrections.

B. State diagram of the model of Section 3

The ephemeral key exchange is changed to use `kemencaps` and `kemdecaps`. The authentication is handled using the same KEM functions. However, KEMTLS does not send `CertificateVerify` messages. Instead, the peer that receives a `Certificate` message sends back a `KEMCiphertext` message with the ciphertext. We modified the relevant rules to implement both server authentication and mutual authentication, and updated the state machine. The new state machine for the KEMTLS handshake is visible in Fig. 2, on the left-hand side. In this figure, the states are the set of variables passed along and modified by the rules that form the transitions between them. The protocol messages exchanged between the client and server state machines are also pictured.

We disabled the PSK and session ticket features of the original model.

C. Performance of the model of Section 3

We ran the model on a server that has two 20-core Intel Xeon Gold 6230 CPUs, which after hyperthreading gives us 80 threads. The server has 192 GB of RAM.

Tamarin runs through all of the lemmas in our proof in 28 hours. We note that communication bottlenecks between cores prevent fully utilising all resources. The model requires 121 GB of RAM to prove all the lemmas, though most individual lemmas need much less memory. In Table 2 we show some of the lemmas that consumed the most time. Note that it is likely possible to prove them in less time, by hiding more “distracting” helper lemmas or writing smarter oracles, but we did not optimise for this.

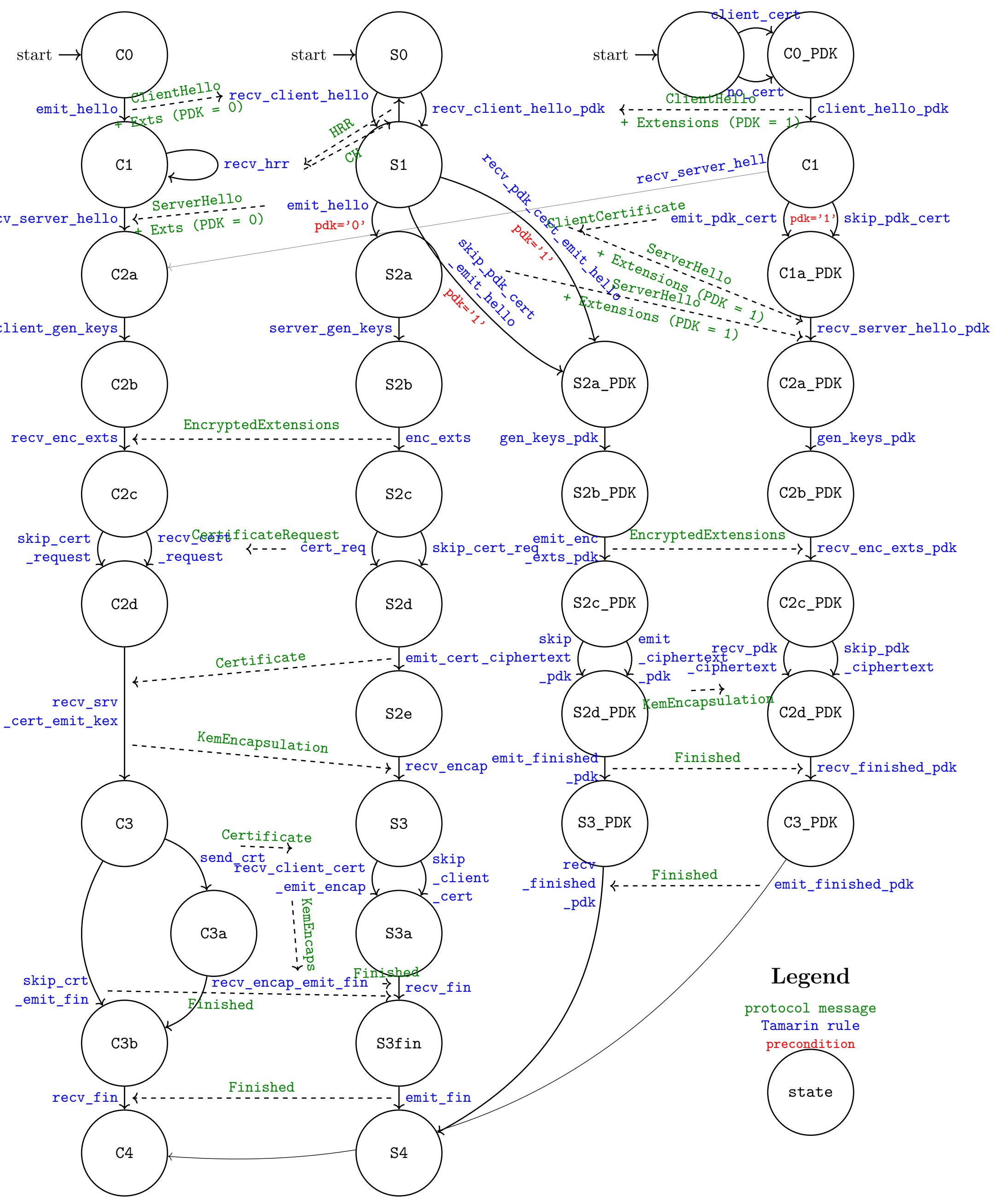

Figure 2: Rules and state transitions in the Section 3 model of KEMTLS and KEMTLS-PDK based on the TLS 1.3 model by [14].

D. State diagram of the model of Section 4

Figure 3 shows the state machine for all rules in this model. As an example, we present a walk-through of the rules associated with an execution of KEMTLS-sauth. This may be

Lemma	Steps	Runtime	Memory
`session_key_auth_agreement`	29116	06:42:01	16 GB
`session_key_agreement`	57680	13:56:04	32 GB
`handshake_secret`	29390	04:40:52	12 GB
`master_secret_pfs`	29535	02:53:11	76 GB

Table 2: Runtime and memory usage of a selection of lemmas from the model described in Section 3

best read in conjunction with Tamarin source code in the files indicated.

- `rule KEMTLS_KEM_s_KeyGen:`[1] Generate a long-term key pair and register it for a server B; output public key and save secret key.
- `rule OCorruptLTK:`[2] Reveal long-term secret key for a particular party.
- `rule ORevealSessionKey:`[2] Reveal the stage key for a selected session and stage.
- `rule KEMTLS_SAUTH_OR_MUTUAL_ClientAction1:`[3] Generate and send the client's first outgoing message (`ClientHello`) in a KEMTLS-sauth or KEMTLS-mutual execution (these have the same first flow). Record action facts for this instance's `Role` and the contributive identifier (`CID`) for stage 1.
- `rule KEMTLS_SAUTH_ServerAction1:`[4] Upon receipt of a `ClientHello`, generate the server's two first outgoing message (`ServerHello`, `ServerCertificate`). Compute stage 1 and 2 keys (CHTS, SHTS). Record action facts for: this instance's `Role`; `Owner`; `Peer` = anonymous; contributive and session identifiers (`CID`, `SID`) for stage 1 and 2 keys; status `Accept` of stage 1 and 2; `ProtocolMode` = `sauth_or_mutual` for stage 1 and 2; the stage keys of stage 1 and 2 (`SK`); and the claimed forward secrecy properties (`FS`) of the stage 1 and 2 keys as of stage 1 and 2 acceptance.
- `rule KEMTLS_SAUTH_ClientAction2:`[5] Using the internal state saved by `KEMTLS_SAUTH_OR_MUTUAL_ClientAction1`, and upon receipt of `ServerHello`, `ServerCertificate`, with `ServerCertificate` containing a public key registered by `KEMTLS_KEM_s_KeyGen` for a server B, decapsulate the ephemeral shared secret, encapsulate against the server's long-term public key, and generate the client's outgoing messages `ClientKEMCiphertext`, `ClientFinished`. Compute stage 3, 4, and 5 keys ($\mathrm{CAHTS}, \mathrm{SAHTS}, \mathrm{CATS}$). Record action facts for: this instance's `Owner` = anonymous; `Peer` = B; contributive and session identifiers (`CID`, `SID`) for

[1] Source code `protocol/kemtls_keygen.spthy`
[2] Source code `model/oracles.spthy`
[3] Source code `protocol/kemtls_sauth_or_mutual_client.spthy`
[4] Source code `protocol/kemtls_sauth_server.spthy`
[5] Source code `protocol/kemtls_sauth_client.spthy`

stage 1–5 keys; status `Accept` of stages 1–5; `ProtocolMode` = `sauth_or_mutual` for stages 1–4 and `sauth` for stage 5; the stage keys of stages 1–5 (`SK`); and the claimed forward secrecy properties (`FS`) of the stage 1–5 keys as of acceptance of stages 3–5.

- `rule KEMTLS_SAUTH_ServerAction2Part1`:[4] This server action rule is split in two: part 1 takes as input `ClientKEMCiphertext`, part 2 takes as input `ClientFinished` but requires that `ClientFinished` be authentic; by splitting the rule we allow the stage 3 and 4 keys to be accepted even if stage 5 rejects due to an invalid `Client-Finished`. Using the internal state saved by `KEMTLS_SAUTH_ServerAction1`, and upon receipt of `ClientKEMCiphertext`, decapsulate using the server's long-term secret key. Compute stage 3 and 4 keys ($\mathrm{CAHTS}, \mathrm{SAHTS}$). Record action facts for: contributive and session identifiers (`CID`, `SID`) for stage 3, 4 keys; status `Accept` of stages 3, 4; `ProtocolMode` = `sauth_or_mutual` for stages 3, 4; the stage keys of stages 3, 4 (`SK`); and the claimed forward secrecy properties (`FS`) of the stage 3, 4 keys as of acceptance of stages 3, 4.

- `rule KEMTLS_SAUTH_ServerAction2Part2`:[4] Using the internal state saved by `KEMTLS_SAUTH_ServerAction2Part1`, and upon receipt of a valid `ClientFinished`, generate the server's outgoing message `ServerFinished`. Compute stage 5 and 6 keys ($\mathrm{CATS}, \mathrm{SATS}$). Record action facts for: contributive and session identifiers (`CID`, `SID`) for stage 5, 6 keys; status `Accept` of stages 5, 6; `ProtocolMode` = `sauth` for stages 1–6; the stage keys of stages 5, 6 (`SK`); and the claimed forward secrecy properties (`FS`) of the stage 1–6 keys as of acceptance of stages 5, 6.

- `rule KEMTLS_SAUTH_ClientAction3`:[5] Using the internal state saved by `KEMTLS_-SAUTH_OR_MUTUAL_ClientAction2`, and upon receipt of a valid `ServerFinished`. Compute stage 6 key (SATS). Record action facts for: contributive and session identifiers (`CID`, `SID`) for stage 6 key; status `Accept` of stage 6; `ProtocolMode` = `sauth` for stages 1–6; the stage key of stage 6 (`SK`); the claimed forward secrecy properties (`FS`) of the stage 1–6 keys as of acceptance of stage 6; and the explicit authentication (`AUTH`) of stages 1–6 as of acceptance of stage 6.

Notice the retroactive upgrading of certain properties such as protocol mode and forward secrecy once later stages accept, for example in `ServerAction2Part2` and `ClientAction3`.

E. Multi-stage security definitions of KEMTLS(-PDK)

Tables 3 and 4 show the instantiation of the forward secrecy parameters for the KEMTLS and KEMTLS-PDK protocol modes under consideration.

- $\mathsf{auth}^C \in \{1, \ldots, M, \infty\}^M$: at which stage each stage in a client session is considered authenticated, or never (∞) (e.g., if $\mathsf{auth}^C[3] = 6$, then stage 3 is considered authenticated once stage 6 accepts;

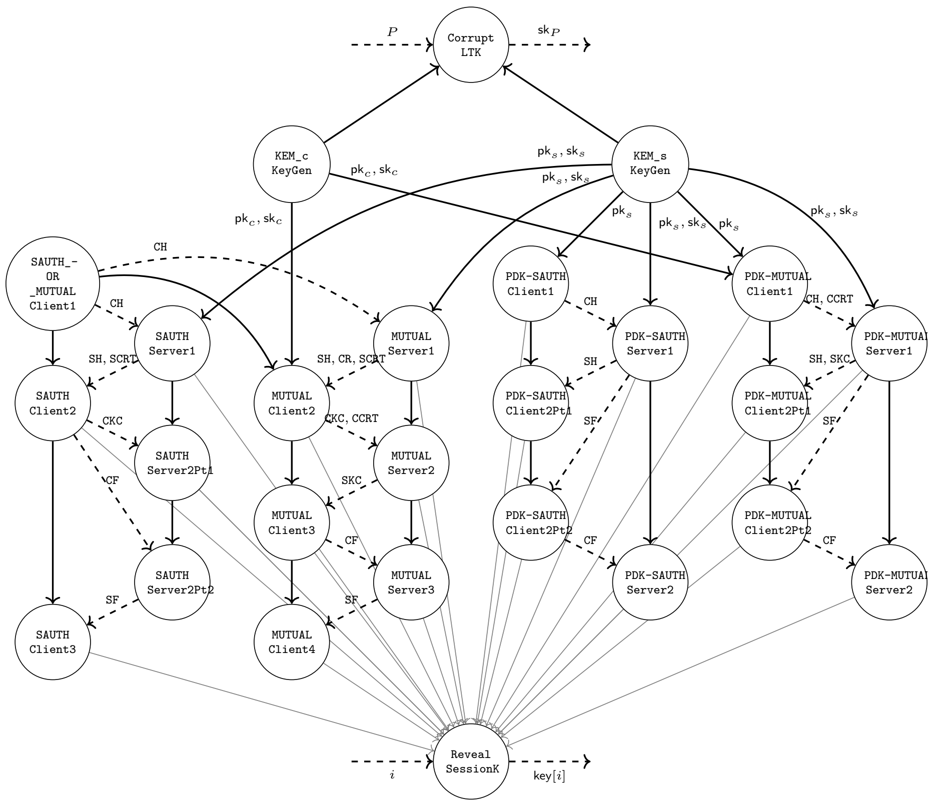

Figure 3: States and transitions for Tamarin KEMTLS model of Section 4.

Legend: Party state: $\longrightarrow$ Input/output message: $\xrightarrow{\texttt{SCRT}}$ Session key: $\longrightarrow$

Parameter	KEMTLS **Server-only auth.**	KEMTLS **Mutual auth.**
auth^C	$(6^{\times 6})$	same as server-only auth.
auth^S	$(\infty^{\times 6})$	$(5^{\times 5}, \infty)$
FS^C	$\begin{pmatrix} \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} \\ & & & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} \\ & & & & \mathsf{wfs2} & \mathsf{fs} \\ & & & & & \mathsf{fs} \end{pmatrix}$	same as server-only auth.
FS^S	$\mathsf{FS}^S_{i,j} = \mathsf{wfs1}$ for all $j \geq i$	$\begin{pmatrix} \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & & \mathsf{wfs1} & \mathsf{fs} & \mathsf{fs} \\ & & & & \mathsf{fs} & \mathsf{fs} \\ & & & & & \mathsf{fs} \end{pmatrix}$
replay	$(\mathsf{nonreplayable}^{\times 6})$	same as server-only auth.

Table 3: Model parameters for KEMTLS

- $\mathsf{auth}^S \in \{1, \ldots, M, \infty\}^M$: similar to auth^C, but for server sessions;
- $\mathsf{FS}^C \in \{0, \mathsf{wfs1}, \mathsf{wfs2}, \mathsf{fs}\}^{M \times M}$: $\mathsf{FS}^C_{i,j}$ is the type of forward secrecy expected in a client session's stage i assuming stage j has accepted;
- $\mathsf{FS}^S \in \{0, \mathsf{wfs1}, \mathsf{wfs2}, \mathsf{fs}\}^{M \times M}$: similar to FS^C, but for server sessions;
- $\mathsf{replay} \in \{\mathsf{nonreplayable}, \mathsf{replayable}\}^M$: whether a stage is expected to be unique against replay attacks or not.

A protocol satisfies the multi-stage security model for particular model parameters as above if, after interacting with honest parties via the oracles above, either:

(a) all tested sessions remain *fresh*, and the adversary correctly guesses the hidden bit b used in the Test oracle; or

(b) some non-replayable stage at an honest party has maliciously accepted, meaning the stage has accepted but there does not exist a unique partner session, and the intended peer's long-term key was not revealed via Corrupt before the stage was considered authenticated as indicated by the auth vector.

The definition of fresh in clause (a) is specialized based on the FS and replay parameters of the model. As per [35, Defn. 2], stage i of a session π is considered fresh if:

1. the stage i key was not revealed (no $\mathsf{Reveal}(\pi, i)$ query); and
2. the stage i key of the partner session, if it exists, was not revealed; and
3. if stage i is replayable: the peer's long-term key was never corrupted (no $\mathsf{Corrupt}(\pi.\mathsf{pid})$);

Parameter	KEMTLS-PDK **Server-only auth.**	KEMTLS-PDK **Mutual auth.**
auth^C	$(4^{\times 4}, \infty)$	$(4^{\times 4}, \infty)$
auth^S	$(\infty^{\times 5})$	$(5^{\times 5})$
FS^C	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs2} & \mathsf{wfs2} & \mathsf{fs} & \mathsf{fs} \\ & & \mathsf{wfs2} & \mathsf{fs} & \mathsf{fs} \\ & & & \mathsf{fs} & \mathsf{fs} \\ & & & & \mathsf{fs} \end{pmatrix}$	same as server-only auth.
FS^S	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} \\ & & & \mathsf{wfs1} & \mathsf{wfs1} \\ & & & & \mathsf{wfs1} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & \mathsf{wfs1} & \mathsf{wfs1} & \mathsf{fs} \\ & & & \mathsf{wfs2} & \mathsf{fs} \\ & & & & \mathsf{fs} \end{pmatrix}$
replay	$(\mathsf{replayable}, \mathsf{nonreplayable}^{\times 4})$	same as server-only auth.

Table 4: Model parameters for KEMTLS-PDK

4. and, based on the expected FS value for this stage:
   a) no forward secrecy: the peer's long-term key was never corrupted;
   b) weak forward secrecy 1: some stage $j \geq i$ of π has accepted and has $\mathsf{FS}_{i,j} = \mathsf{wfs1}$, and π has a contributive partner at stage i;
   c) weak forward secrecy 2: some stage $j \geq i$ of π has accepted and has $\mathsf{FS}_{i,j} = \mathsf{wfs2}$, and either (i) π has a contributive partner at stage i or (ii) the peer's long-term key was never corrupted;
   d) forward secrecy: some stage $j \geq i$ of π has accepted and has $\mathsf{FS}_{i,j} = \mathsf{fs}$, and either (i) π has a contributive partner at stage i or (ii) the peer's long-term key was not corrupted before π accepted stage j.

F. Performance of the model of Section 4

Table 5 shows the run-time for the various lemmas, for each KEMTLS variant on its own, and when all four KEMTLS variants are run simultaneously Tamarin was restricted to using 16 cores, and the times shown are wall-clock time. Total CPU time will be greater than the wall-clock time, but typically less than 16× wall-clock time since Tamarin hits communication bottlenecks preventing it from loading all cores to 100%. Results are measured on the same system as in Appendix C, with `tamarin-prover` version 1.16.1.

Appendix C clearly shows that the mutual versions of the protocols require a bit more work for Tamarin than the server-authenticated protocols. Interestingly, this difference is more pronounced for KEMTLS than it is for KEMTLS-PDK. This might be because in KEMTLS, mutual authentication requires the exchange of significantly more messages. We also see how deniability for mutual authentication is much harder to prove than

Lemma	KEMTLS			KEMTLS-PDK			**All 4**
	sauth	mutual	both	sauth	mutual	both	**variants**
`reachable_*`	01:17	01:20	04:32	01:46	01:36	04:40	13:25
`attacker_works_*`	00:17	00:46	01:16	00:17	00:23	00:53	12:04
`match_*`	01:02	01:22	02:55	00:55	01:14	02:46	09:53
`sk_sec_nofs_client`	00:05	00:07	00:16	00:05	00:05	00:14	00:41
`sk_sec_nofs_server`	00:05	00:06	00:12	00:05	00:06	00:14	00:40
`sk_sec_wfs1`	00:21	00:10	01:05	00:17	00:18	00:41	03:00
`sk_sec_wfs2`	00:36	00:28	01:30	00:28	00:22	01:23	24:28
`sk_sec_fs`	01:20	03:05	06:38	01:21	01:33	05:07	1:39:58
`malicious_accept.`	00:13	01:40	04:13	00:17	00:22	01:39	27:29:37
`deniability` (abbr.)	01:02	12:15	—	00:24	29:10	—	—
Total (excl. `den.`)	05:16	09:05	22:38	05:30	06:00	17:38	30:13:46

Table 5: Wall-clock run-time (hh:mm:ss) for Tamarin proofs of lemmas from Section 4

deniability of unilaterally authenticated KEMTLS(-PDK). Finally, proving the security properties for more than a single protocol has a large effect on the runtime, as we expect given the opportunities for cross-protocol interaction.