A Comprehensive Symbolic Analysis of TLS 1.3

Cas Cremers
University of Oxford, UK

Marko Horvat
MPI-SWS, Germany

Jonathan Hoyland
Royal Holloway, University of London, UK

Sam Scott
Royal Holloway, University of London, UK

Thyla van der Merwe
Royal Holloway, University of London, UK

ABSTRACT

The TLS protocol is intended to enable secure end-to-end communication over insecure networks, including the Internet. Unfortunately, this goal has been thwarted a number of times throughout the protocol's tumultuous lifetime, resulting in the need for a new version of the protocol, namely TLS 1.3. Over the past three years, in an unprecedented joint design effort with the academic community, the TLS Working Group has been working tirelessly to enhance the security of TLS.

We further this effort by constructing the most comprehensive, faithful, and modular symbolic model of the TLS 1.3 draft 21 release candidate, and use the Tamarin prover to verify the claimed TLS 1.3 security requirements, as laid out in draft 21 of the specification. In particular, our model covers *all* handshake modes of TLS 1.3.

Our analysis reveals an unexpected behaviour, which we expect will inhibit strong authentication guarantees in some implementations of the protocol. In contrast to previous models, we provide a novel way of making the relation between the TLS specification and our model explicit: we provide a fully annotated version of the specification that clarifies what protocol elements we modelled, and precisely how we modelled these elements. We anticipate this model artifact to be of great benefit to the academic community and the TLS Working Group alike.

KEYWORDS

symbolic verification, authenticated key exchange, TLS 1.3

1 INTRODUCTION

The Transport Layer Security (TLS) protocol is the *de facto* means for securing communications on the World Wide Web. Initially released as Secure Sockets Layer (SSL) by Netscape Communications in 1995, the protocol has been subject to a number of version upgrades over the course of its 20-year lifespan. Rebranded as TLS when it fell under the auspices of the Internet Engineering Task Force (IETF) in the mid-nineties, the protocol has been incrementally modified and extended. In the case of TLS 1.2 and below, these modifications have taken place in a largely retroactive fashion; following the announcement of an attack [6, 7, 18, 20, 32, 43, 49], the TLS Working Group (WG) would either respond by releasing a protocol extension (A Request for Comments (RFC) intended to provide increased functionality and/or security enhancements) or by applying the appropriate "patch" to the next version of the protocol. For a more detailed analysis of the development and standardisation of TLS see [45].

Prior to the announcement of the BEAST [26] and CRIME [27] attacks of 2011 and 2012, respectively, such a strategy was valid given the frequency with which versions were updated, and the limited number of practical attacks against the protocol.

Post-2011, however, the heightened interest in the protocol and the resulting flood of increasingly practical attacks against it [1–3, 5, 9, 13, 15, 16, 26, 27, 29, 31, 41, 42, 44] rendered this design philosophy inadequate. Coupled with pressure to increase the protocol's efficiency (owing to the release of Google's QUIC Crypto [37]), the IETF started drafting the next version of the protocol, TLS 1.3, in the Spring of 2014. Unlike the development of TLS 1.2 and below, the TLS WG adopted an "analysis-prior-to-deployment" design philosophy, welcoming contributions from the academic community before official release. There have been substantial efforts from the academic community in the areas of program verification– analysing implementations of TLS [12, 14], the development of computational models– analysing TLS within Bellare-Rogaway style frameworks [24, 25, 28, 33, 35, 38], and the use of formal methods tools such as ProVerif[17] and Tamarin[48] to analyse symbolic models of TLS [4, 10, 22, 30]. All of these endeavours have helped to both find weaknesses in the protocol and confirm and guide the design decisions of the TLS WG.

The TLS 1.3 draft specification however, has been a rapidly moving target, with large changes being effected in a fairly regular fashion. This has often rendered much of the analysis work 'outdated' within the space of few months as large changes to the specification effectively result in a new protocol, requiring a new wave of analysis.

In this work we contribute to what is hopefully the last wave of analysis of TLS 1.3 prior to its official release. We present a tool-supported, symbolic verification of a near-final draft of TLS 1.3, adding to the large effort by the TLS community to ensure that TLS 1.3 is free of the many weaknesses affecting earlier versions, and that it is imbued with security guarantees befitting such a critical protocol. We note that most of the cryptographic mechanisms in the current TLS 1.3 draft are stable, and other than fluctuations

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134063

surrounding the zero-Round-Trip-Time (0-RTT) mechanism [40], we do not expect substantial changes to come.

1.1 Contributions

Our main contributions in this work are as follows:

(1) We develop a symbolic model of the latest specification of TLS 1.3 (draft 21) that considers all the possible interactions of the available handshake modes, including PSK-based resumption and 0-RTT. Its fine-grained, modular structure greatly extends and refines the coverage of previous symbolic models that were successfully used to discover sophisticated interaction attacks, including that of Cremers et al. [22]. Our model effectively captures a new TLS 1.3 protocol, incorporating the many changes that have been made to the protocol since the development of these previous models. We also note that our model is highly flexible and can easily accommodate the removal of the 0-RTT mechanism, should the need arise.
(2) We prove the majority of the specified security requirements of TLS 1.3, including the secrecy of session keys, perfect forward secrecy (PFS) of session keys (where applicable), peer authentication, and key compromise impersonation resistance. We also show that after a successful handshake the client and server agree session keys and that session keys are unique across handshakes.
(3) We uncover a behaviour that may lead to security problems in applications that assume that TLS 1.3 provides strong authentication guarantees.
(4) We provide a novel way of exhibiting the relation between the specification and our model: we provide an annotated version of the TLS 1.3 specification that clarifies which parts are modelled and how, and which parts were abstracted. This provides an unprecedented level of modelling transparency and enables a straightforward assessment of the faithfulness and coverage of our model. We anticipate that this output will be of great benefit to the academic community analysing TLS 1.3, as well as the TLS Working Group as it provides a clear and easy-to-understand mapping between the TLS 1.3 specification and a TLS 1.3 model.

All our Tamarin input files, proofs, and the annotated TLS 1.3 specification that shows the relation between the RFC and the model, can be downloaded from [21].

1.2 Related work

As mentioned, there has been a great deal of work conducted in the complementary analysis spheres pertinent to TLS 1.3. Of most interest to this work are the symbolic analyses presented in [22], [4] and [10].

The work in [22] by Cremers et al. offered a symbolic model and accompanying analysis of draft 10 of the TLS 1.3 specification, using the Tamarin prover. Since then, there have been multiple changes made to the specification - 10 drafts worth of changes to be precise. These updates have included major revisions of the 0-RTT mechanism and the key derivation schedule. In draft 10, the sending of early data required a client to possess a semi-static (EC)DH value of the server. This particular handshake mode was removed and replaced by a pre-shared key (PSK) 0-RTT handshake mode– early data can now only be encrypted using a PSK. In fact, the PSK mechanism has been greatly enhanced since draft 10 with new PSK variants and binding values being incorporated in to the specification. Post-handshake authentication was officially incorporated from draft 11 onwards and a few drafts later, post-handshake authentication was enabled to operate with the PSK handshake mode. Another change to be incorporated after draft 10 was the inclusion of 0.5-RTT data - the server being able to send fully protected application data as part of its first flight of messages.

All of these changes have resulted in what is effectively a very different TLS 1.3 protocol, particularly from a symbolic perspective. As a Tamarin model aims to consider the interaction of all possible handshake modes and variants, changes to these modes, as well as the inclusion of new post-handshake combinations, results in a very different different set of traces to be considered when proving security properties. Hence, this work presents a substantially different model to [22], and follows a far more fine-grained and flexible approach to modelling TLS 1.3.

The work in [4] is an analysis of TLS 1.3 by the Cryptographic protocol Evaluation towards Long-Lived Outstanding Security (CELLOS) Consortium using the ProVerif tool. Announced on the TLS WG mailing list at the start of 2016, it showed the initial (EC)DHE handshake of draft 11 to be secure in the symbolic setting. In comparison to our work, this analysis covers only one handshake mode of a draft that is now somewhat outdated.

The ProVerif models of draft 18 presented by Bhargavan et al. in [10] include most TLS 1.3 modes, and cover rich threat models by considering downgrade attacks (both with weak crypto and downgrade to TLS 1.2). However, unlike our work, they do not consider all modes, as they do not consider the post-handshake client authentication mode. While they cover relative strong authentication guarantees (which led to the discovery of an unknown key share attack), their analysis did not uncover the potential mismatch between client and server view that we describe in Section 5.2.

1.3 Paper organisation

The paper is organised as follows. In Section 2 we describe the TLS 1.3 protocol and the security properties claimed in the specification. Section 3 describes our Tamarin model and provides a few Tamarin prover fundamentals. In Section 4, we describe our encoding of the security guarantees, followed by Section 5 where we describe our results. Section 6 covers the relationship between our model and the specification document, discussing how we provide a website that describes our model side-by-side with the specification, giving us unprecedented modelling transparency. We conclude in Section 7.

2 TLS 1.3

In this section we provide a brief description of the TLS 1.3 protocol as is necessary for understanding our symbolic model, and we outline the claimed security properties and guarantees of the protocol.

2.1 New mechanisms

The three years of effort that has gone into crafting and fine-tuning both the security and efficiency mechanisms of TLS 1.3 is readily apparent in the large structural departures from TLS 1.2. The two protocols have broadly similar goals but exhibit many differences. For example, a full TLS 1.3 handshake requires one fewer round trip before a client can transmit protected application data, and the new zero round trip time (0-RTT) mechanism allows less sensitive application data to be sent by the client as part of its first flight of messages.

TLS 1.3 has three key exchange modes, namely, Diffie–Hellman exchange (DHE), pre-shared key (PSK) exchange, and PSK coupled with DHE. These modes enable useful features like session resumption and the transmission of early application data. Additionally, there are a number of handshake variants that allow for group renegotiation and the sending of context-dependent, optional messages. Each of these variants has different properties and offers different security guarantees.

Furthermore, TLS 1.3 has three post-handshake mechanisms covering traffic key updates, post-handshake client authentication, and the sending of new session tickets (NSTs) for subsequent resumption via a PSK. The handshake protocol maintains a rolling transcript, on which both parties must agree. This transcript takes the form of a hash value of all of the handshake messages. Post-handshake messages, however, are not included in this transcript resulting in different security properties for the post-handshake mechanisms.

We analyse all of the TLS 1.3 key exchange modes, handshake variants, and post-handshake mechanisms simultaneously, considering all possible interactions between them. We provide a brief description of these components as well as associated message flow diagrams.

2.1.1 Diffie–Hellman exchange (DHE). The default mode of TLS 1.3 allows for ephemeral Diffie–Hellman (DH) keys to be established either over a finite field or using elliptic curves.

In an initial (EC)DHE handshake, as depicted in Figure 1, the client sends a `ClientHello` message containing a random nonce, i.e. a freshly generated random value, and a list of symmetric algorithms. The client also sends a set of DH key shares and the associated groups, `KeyShare`, and potentially some other extensions.

Upon receipt of a `ClientHello` message, the server selects appropriate cryptographic parameters for the connection and responds with a `ServerHello` message. This message contains a server-generated random nonce, an indication of the selected parameters and potentially some other extensions. The server also sends a `KeyShare` message, along with an `EncryptedExtensions` message and optionally a `CertificateRequest` message.

The `KeyShare` contains the server's choice of group and its ephemeral DH key share. The client and server key shares are used to compute handshake and application traffic keys.
The `EncryptedExtensions` message contains material that is not necessary for determining cryptographic parameters. For instance, the draft specification lists the server name and the maximum TLS fragment length as possible values to be sent in this message. The

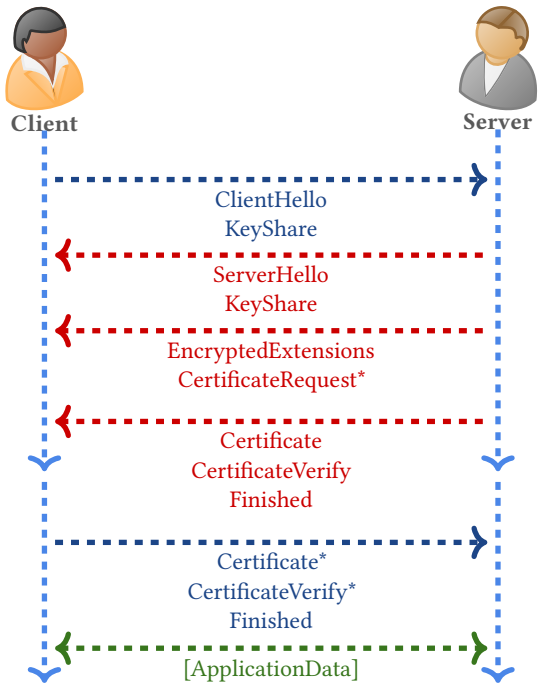

Figure 1: A full TLS 1.3 handshake (Section 2.1.1)

`CertificateRequest` message indicates that the server requests client authentication in the mutual authentication case.

The server will also send a `Certificate` message, containing the server's certificate and a `CertificateVerify` message, which is a digital signature over the current transcript. These two messages allow the client to authenticate server. The server also sends a `Finished` message. This message is a Message Authentication Code (MAC) over the entire handshake, providing key confirmation and binding the server's identity to the computed traffic keys.

The client responds with `Certificate` and `CertificateVerify` messages, if requested, and then sends its own `Finished` message. These message flows are depicted in Figure 1.

2.1.2 Pre-shared key (PSK). In the event that a PSK has been established, a client and a server can begin communicating without a DH exchange. This is potentially attractive for low-power environments, however, without a DHE the connection loses perfect forward secrecy (PFS). In a PSK handshake, the server authenticates via a PSK.

2.1.3 PSK with DHE. By combining a PSK with DHE this mode maintains PFS whilst limiting the number of expensive public key operations that the server needs to perform.

2.1.4 Group renegotiation. It can be the case that the groups sent by a client are not acceptable to the server. In this case, the server may respond with a `HelloRetryRequest` message. This indicates to the client which groups the server will accept, and provides the client with the opportunity to respond with an appropriate key share before returning to the main handshake.

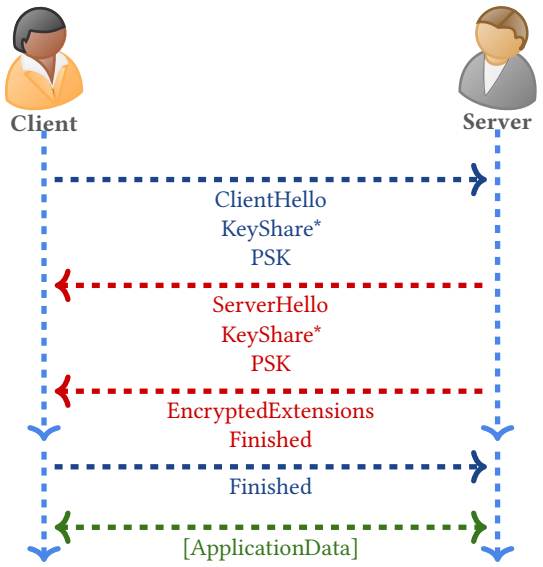

Figure 2: A PSK resumption handshake (Section 2.1.7)

2.1.5 New session ticket (NST). After a successful handshake, the server can issue an NST at any time. These tickets create a binding to a resumption-specific secret and can be used by the client as PSKs in subsequent handshakes.

2.1.6 PSK binder. A PSK binder is a value that binds a PSK to the handshake where the PSK is offered by a client in a `ClientHello` message and, if the PSK was generated by a server in-band, to the handshake where it was generated. A `ClientHello` can contain multiple binders arranged in a list, where each binder is computed over a hash of the `ClientHello` message (without the binder list itself).

2.1.7 Session resumption and PSK. This handshake variant allows a client to use a key established out-of-band (OOB) to start a new session, or to use an NST established in a previous handshake to resume the session. This avoids the use of expensive public-key operations and in the case of a resumption, ties the security context of the new connection to the original connection. Note that a server may reject a resumption attempt made by a client, so the specification recommends that the client supplies an additional (EC)DHE key share with its PSK when trying to resume a session. Figure 2 depicts a PSK resumption handshake.

2.1.8 Zero round trip time (0-RTT). A client can use a PSK to send application data in its first flight of messages, reducing the latency of the connection. As noted in the TLS 1.3 draft specification, this data is not protected against replay attacks. If the communicating entities wish to take advantage of the 0-RTT mechanism, they should provide their own replay protection at the application layer. A 0-RTT handshake is depicted in Figure 3.

2.1.9 Post-handshake client authentication. After a successful handshake, the server can send a `CertificateRequest` message. If the client responds with an acceptable certificate, then the server might authenticate the client. However, because the specification allows certificates to be rejected 'silently', the client cannot be sure of its authentication status in general. We discuss this in greater detail in Section 5.2.

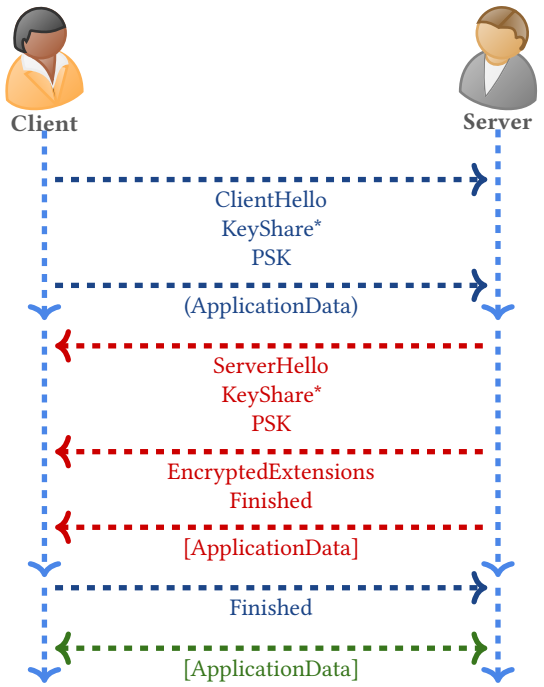

Figure 3: A 0-RTT handshake (Section 2.1.8)

2.1.10 Key update. After a successful handshake, either party can request an application data key update. Because the read and write keys for application data are independent, either party can immediately update their write key after requesting a key update.

2.1.11 Key derivation. A TLS 1.3 handshake will generate a set of keys on which both the client and server agree. The specification defines a key schedule which uses the repeated application of an HMAC-based key derivation function (HKDF) [34] to combine the secret inputs with fixed labels so as to generate a set of independent keys.

The key schedule has two secret inputs, the (EC)DHE and the PSK. Depending on the handshake mode, either one or both of these will be used. The key schedule also includes the transcript hash in the key derivation. Because the transcript includes nonces, even if the secret inputs are repeated, the generated keys are guaranteed to be independent.

2.2 Stated goals and security properties

The TLS 1.3 handshake protocol is intended to negotiate cryptographic keys via the mechanism of authenticated key exchange (AKE). These keys can then be used by the record layer to provide critical security guarantees, including confidentiality and integrity of messages. As stated in Section 2.1, TLS 1.3 makes use of independent keys to protect handshake messages and application data

messages: protection of the handshake messages starts with the server's `EncryptedExtensions` message, and in the majority of handshake modes, protection of application data messages occurs after the transmission of the server and client `Finished` messages, respectively. In the case of a 0-RTT handshake, application data is protected with a PSK as part of the client's first flight of messages.

The TLS specification[46, Appendix E.1] lists eight properties that the handshake protocol is required to satisfy:

1. **Establishing the same session keys**. Upon completion of the handshake, the client and the server should have established a set of session keys on which they both agree.
2. **Secrecy of the session keys**. Upon completion of the handshake, the client and server should have established a set of session keys which are known to the client and the server only.
3. **Peer authentication**. In the unilateral case, upon completion of the handshake, if a client `C` believes it is communicating with a server `S`, then it is indeed `S` who is indeed executing the server role. An analogous property for the server holds in the bilateral (mutual) authentication case.
4. **Uniqueness of session keys**. Each run of the protocol should produce distinct, independent session keys.
5. **Downgrade protection**. An active attacker should not be able to force the client and the server to employ weak cipher suites, or older versions of the TLS protocol.
6. **Perfect Forward Secrecy (PFS)**. In the case of compromise of either party's long-term key, sessions completed before the compromise should remain secure. This property is not claimed to hold in the PSK key exchange mode.
7. **Key compromise impersonation (KCI) resistance**. Should an attacker compromise the long-term key of party `A`, the attacker should not be able to use this key to impersonate an uncompromised party in communication with `A`.
8. **Protection of endpoint identities**. The identity of the server cannot be revealed by a passive attacker that observes the handshake, and the identity of the client cannot be revealed even by an active attacker that is capable of tampering with the communication.

We model six out of the eight required properties, omitting downgrade protection and the protection of endpoint identities. Also, as stated previously, 0-RTT mechanisms allow for replay of early data across sessions. We discuss the reduced 0-RTT security properties as well as the properties described above more fully in Section 4.

The draft specification refers to RFC 3552[47] for an informal description of the TLS 1.3 threat model. This model assumes a Dolev-Yao attacker[23]– an attacker that can perform man-in-the-middle (MITM) attacks by being able to replay, insert, delete, and modify messages at will. We consider a strictly more powerful attacker, as we will explain in Section 4.1.

3 MODELLING THE PROTOCOL

3.1 The Tamarin prover

The Tamarin prover [48] is a symbolic modelling and analysis tool for security protocols. Its specification language facilitates the construction of highly detailed models of security protocols, their security requirements and powerful Dolev-Yao-style attackers. The verification algorithm of Tamarin is based on constraint solving and multiset-rewriting techniques, which allows its users to prove intricate security properties in complex protocols exhibiting branches and loops. Moreover, it offers state-of-the-art symbolic Diffie-Hellman support. Tamarin inherently supports non-monotonic state and it includes an extensive graphical user interface that enables the visualisation and interactive construction of proofs.

These features make Tamarin a good fit for the modelling and in-depth analysis of highly complex protocols such as TLS 1.3. In particular, the support for branching allowed us to model the decisions that the protocol participants can make during execution, the loops were instrumental in covering repeated connections within a single session, and the main security aspects of TLS 1.3 critically depend on Diffie-Hellman key exchange. The non-monotonic state support enabled us to model branching without having to resort to custom-tailored hacks or having to rely on the considerable over-approximation where all branches can be considered simultaneously. Lastly, the visualisations of attacks found by Tamarin provided us with a way to quickly identify potential problems, with either the protocol or our model– the graphical user interface was a great asset in guiding our TLS 1.3 verification workflow.

We introduce the Tamarin specification language by considering a toy example rule:

```
rule Example_Rule:
 [ !Key(x,y), Fr(n) ]--[ Send(x,n,y) ]->[ Out(senc{n}y) ]
```

Tamarin rules consist of three respective parts: a *left-hand side*, *actions* and a *right-hand side*. The rules are used to define a transition system, whose global state is maintained as a multiset of so-called *facts*. The initial state of the transition system is the empty multiset. A rule can only be executed if all the facts on its left-hand side are available in the current state. When a rule is executed, it will *consume* the facts on the left, i.e. removing them from the global state, and *produce* facts on the right, i.e. adding them to the global state. Facts are either linear or persistent; each available linear fact can only be consumed once, and persistent facts can be consumed any number of times. Actions specify observable events in every trace, and are used to express security properties. There are several special facts, one of which is the `Fr` fact, which is implicitly available in all states because of Tamarin's built-in `Fresh` rule, and is used to produce fresh (unique) values. In the above example, if an agent `x` owns a symmetric key `y`, then it can send out a fresh value `n` that is symmetrically encrypted with `y`. A basic property that states the secrecy of `n` can be encoded as a simple lemma:

```
lemma Example_Property:
 "All x y n #i. Send(x,n,y) @ i ==> not Ex #j. K(n) @ j"
```

A more realistic lemma needs to account for the attacker model. For example, in the presence of an attacker who can compromise symmetric keys, a formula resembling `not Revealed(y) @ k` needs to be conjoined with the left-hand side of the above property or it will be trivially false.

We defer the details of our Tamarin model to Section 4, and note differences to other TLS 1.3 models in the next section.

3.2 A comprehensive model

Using Tamarin's modelling framework we devised a comprehensive symbolic model of TLS 1.3 that captures the specified protocol behaviours, as well as unexpected behaviours that arise from a complex interaction of an unbounded number of sessions. Our model captures these behaviours in the presence of a powerful attacker[1].

Other TLS 1.3 analyses consider the constituent parts of TLS 1.3, viewing these as separate protocols, and proceed to tie the individual proofs together with a compositionality result. For instance, [10] considers the resumption mechanism as a separate protocol in which both the client and the server take as input a symmetric value—the PSK. If the PSK remains unknown to the attacker in every execution of the resumption protocol, a gap remains to be filled before concluding that the full handshake always completes without the attacker knowing the PSK. This gap is filled by a manual compositionality proof. In our work, there is no need for such manual proofs; composition is trivially satisfied by our comprehensive model, as Tamarin considers all the possible interactions in proving each property.

Although our model undoubtedly draws from the Tamarin models described in [30] and [22], we opted to model TLS 1.3 with a significant increase in fidelity to the draft specification. Such an approach resulted in an improved ability to capture the full functionality of TLS, as well as a broader class of realistic attacks. This class includes the coverage of complicated interaction attacks, such as the post-handshake client authentication attack in [22]. Additionally, by closely matching our model to the specification and allowing for an almost line-per-line comparison, we achieve full transparency regarding which parts of the specification we abstract away from, and which assumptions our modelling process relies on. We discuss the relation between our model and the RFC in detail in Section 6.

Not only is our model more comprehensive than the Tamarin models that precede it, it also incorporates the many changes to the TLS 1.3 specification that have materialised since the development of these models. In the following sections, we describe our modelling process, pointing out enhancements over the previous models.

3.3 Closely modelling the specification

As with previous models [22], we employ the use of Tamarin rules to model state transitions within the TLS 1.3 protocol. However, our state transitions are far more fine-grained and modular in comparison to [22], modelling the effective change in state as a result of transmission, receipt and processing of cryptographic parameters. For instance, a basic, initial TLS 1.3 handshake invokes up to 21 different rules and the associated state transitions before post-handshake operations can commence. These state transitions are depicted in Figure 4, and correspond to message flights an cryptographic processing as described in Section 2.1, Figure 1. The full state diagram can be found in Appendix A.

Below we provide an example of one of our rules:

```
rule send:
[ SendStream(~tid, $actor, $peer, auth_status, app_key_out),
  Fr(~data)
]
--[ Send(~tid),
    SendData(~tid, $actor, $peer, auth_status, ~data)
]->
[ SendStream(~tid, $actor, $peer, auth_status, app_key_out),
  Out(senc{data_record(~data)}app_key_out)
]
```

We also note the extensive use of macros in our model, which is enabled by the `m4` preprocessor and allowed us to cover most of the specification, whilst syntactically keeping our model close to it. For example, our `ClientHello` message is a macro that expands to:

```
handshake_record('1',
    ProtocolVersion,
    ClientRandom,
    '0', // legacy_session_id
    $cipher_suites,
    '0', // legacy_compression_methods
    ClientHelloExtensions)
```

which reflects almost exactly how it is written in our Tamarin files. `ClientRandom` is itself another macro, defined to be the value of the client nonce `nc`. In Tamarin's syntax, constants are enclosed by single quotes. Constructing the model in this fashion enables a direct syntactic comparison to the specification. Previous Tamarin models also employ macros, but the connection to the specification is much less evident. For instance, in [22] `ClientHello` is defined to be the pair of values `nc,pc`, representing the client's nonce and "parameters", which serves as a placeholder for handshake values that are abstracted away.

In our model we have tried to define cryptographic components in a way that is reminiscent of imperative programming. As in the specification, we compute the handshake secret by computing the function `HKDF-Extract(gxy,es)`, and the handshake keys are computed by applying a `Derive-Secret` function to this value. This is not strictly necessary due to the assumption of perfect cryptography, but it makes it easier to connect our model to the specification.

3.4 Advanced features

In our model we capture a number of complicated interactions and logic flows inherent to the TLS 1.3 handshake, greatly improving on preceding models, adding features to the model which we consider to be 'advanced'.

Group negotiation. We model the client and the server as having a limited ability to negotiate the group used in the Diffie–Hellman key exchange.

In Tamarin, any value can be used as a group generator. Typically, the fixed (public) constant `'g'` is used, which represents all parties agreeing to use a single group ahead of time. On receiving a key share and storing it in the variable `gx`, we simulate checking that the element resides in this group by pattern matching the value as `'g'^x = gx`. Intuitively, this corresponds to checking that $\exists x \,.\, g^x = gx$.

In Tamarin's syntax, variables that are always instantiated with public values are prefixed by `$`. In our model, the client starts with a pair of public values `$g1`, `$g2` that represent two supported groups,

[1] We defer discussion of our attacker capabilities to Section 4.1.

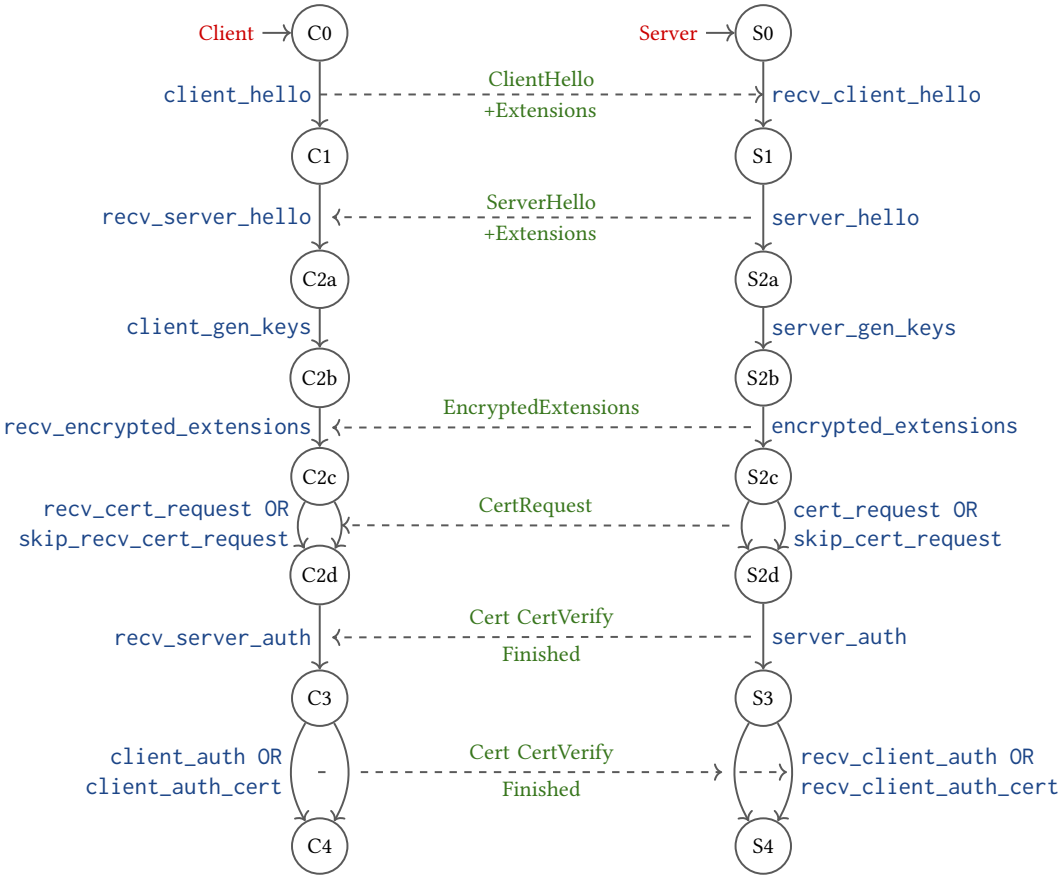

Figure 4: Partial state diagram for full TLS 1.3 handshake. Tamarin rules are indicated in blue. The messages exchanged between entities are given in green. Our full model contains many more transitions. We omit these here for the sake of simplicity.

and offers these to the server along with a corresponding key share for `$g1`. Similarly, the server starts with a supported group `$g`. The model allows the server to return a `HelloRetryRequest` to the client, enforcing that `$g` is not equal to `$g1`, and expects the client to return instead a key share that matches `$g2`.

This interaction enables a much greater coverage of DH key exchange with respect to previous models, and opens up the possibility of future extensions to this work. One such extension would be to model a weak group by permitting the attacker to reveal the corresponding DH exponents.

Handshake flows. One of the most complex elements inherent to modelling TLS 1.3 is the vast number of possible state machine transitions. After a session resumption, the server can choose between using the PSK only, or using the PSK along with a DH key share. Alternatively, the server might reject the PSK entirely, and fall back to a regular handshake, or request that the client use a different group for the DH exchange. Additionally, there are several complex messages that can be sent in the post-handshake state: client authentication requests, new session tickets, and key update requests.

Since all of the above interactions can happen asynchronously, the resulting model becomes very complex and requires sophisticated handling logic. A number of complicated protocol flows, involving any number of sequential handshake modes and post-handshake extensions can, and will, transpire and we deal with this eventuality by modelling all possible handshake modes in a very modular fashion. Other models are, by and large, not capable of capturing complicated protocol flows.

4 ENCODING THE THREAT MODEL AND THE SECURITY PROPERTIES

4.1 Threat Model

We consider an extension of the Dolev-Yao (DY) attacker [23] as our threat model. The DY attacker has complete control of the network, and can intercept, send, replay, and delete any message. To construct a new message, the attacker can combine any information previously learnt, e.g., decrypting messages for which it knows the key, or creating its own encrypted messages. We assume perfect cryptography, which implies that the attacker cannot encrypt, decrypt or sign messages without knowledge of the appropriate keys.

In order to consider different types of compromise, we additionally allow the attacker to do the following:

- compromise the long-term keys of protocol participants,
- compromise their pre-shared keys, whether created OOB or through a NST, and
- compromise their DH values.

Note that TLS 1.3 is not intended to be secure under the full combination of all these types of compromise. For example, session key secrecy can be broken by an attacker who eavesdrops on the communication and compromises the DH values of a single protocol participant.

A natural approach is to weaken the attacker model by adding realistic constraints until either the claimed security goals of the protocol are achieved, or the corresponding attackers become weaker than the ones we expect to face in practice. This workflow requires us to express, with high granularity, exactly what needs to be protected and when each of the claimed TLS 1.3 properties can be expected to hold.

We now give our formal definitions of the TLS 1.3 security properties mentioned in Section 2, noting where each property is covered in our model.

4.2 Security properties

We encode the claimed security properties of TLS 1.3 as lemmas in the specification language of Tamarin. Here we discuss the relationship between the lemmas we prove in the model, and the desired properties in the specification.We note that there is some overlap between the material in the stated goals expressed in Section 2.2. For example, the requirement for PFS is effectively a modifier to the requirement for secret session keys. Where possible, we will prove these properties via distinct lemmas to aid in the comprehension of the model. However, it is also possible to combine many of the properties into a single, more complex lemma.

4.2.1 Establishing the same session keys. The definition of this first property is taken from [19], where it is referred to as a consistency property. However, there is ambiguity in the circumstances that are necessary and sufficient for two protocol participants to establish the same keys. An answer to this question is typically given through the well-established practice of defining *session partnering* [8, 19, 36]. One possible way to do so is to assign session identifiers in terms of a value (or pair of values) on which the two parties agree. We opted for the least restrictive session identifier, namely the pair of nonces generated by the client and the server. Therefore, if a *partnered* client and server complete the handshake, then they must agree on session keys.

We consider this property with respect to an attacker that can compromise all session keys except for those that are identical to that of the test session, i.e. the session in which the attacker attempts to obtain information about the key [19, 36]. This property is captured by our lemma `session_key_agreement`.

4.2.2 Secrecy of the session keys. The `secret_session_keys` lemma is used to prove property (2) in Section 2.2.

The `secret_session_keys` lemma we prove appears in full detail in Figure 5. The intuition for this lemma is that if an actor believes it has established a session key with an authenticated peer, then the attacker does not know the key. However, given the capabilities of the attacker, this will not hold without imposing some restrictions. This is why the additional clauses are required.

The five conditions stated in the depicted lemma are generally repeated across all lemmas, and encapsulate the basic assumption we make about our attacker. We describe them in more detail here: The first imposes the restriction that the long-term signing key of the peer is not compromised[2]. This restriction can additionally be understood to signify that the actor is communicating with an honest peer, since the attacker can effectively simulate a party when in possession of its long-term key. Furthermore, it should be noted that the attacker is still allowed to compromise the peer's long-term key (LTK) *after* the session key is established. Hence we show that the session keys achieve PFS with respect to the LTK.

The second and third clauses bar the attacker from revealing any DH exponents generated by the client or the server from before the session key was established. The attacker may reveal exponents that are generated after the session key is established.

The last two clauses specify that the attacker cannot compromise a PSK associated with either the actor or the peer. Note that the attacker is restricted from revealing these PSKs even after the session key has been established, which corresponds to the proviso in the specification that the PSK-only exchange mode does *not* provide PFS. We discuss this in more detail in Section 4.2.6.

4.2.3 Peer Authentication. The specification defines this property somewhat informally, as a form of authentication whereby both parties should agree on the identity of their peer. Looking at this more formally through the lens of Lowe's hierarchy of authentication[39], this definition corresponds to weak agreement. In particular, we note that this does not imply recentness—the requirement that the peer is currently running the protocol—nor does it specify whether any other values should be agreed upon.

We initially model this property via our `entity_authentication` lemma. Entity authentication is modelled in two parts so as to capture the distinction between the bilateral (mutual) and unilateral authentication cases. Authentication in the unilateral case means that if a client completes a TLS handshake, apparently with a server, then the server previously ran a TLS handshake with the client, and they both agree on certain data values of the handshake, including the identity of the server and the nonces used. Note that this is already a stronger property than is stipulated in the specification. Here we prove non-injective agreement on the nonces, which additionally provides recentness since both parties contribute a fresh nonce to the handshake. The unilateral entity authentication lemma we prove appears in Figure 6.

The intuition for this lemma is that if a client believes it has agreed on a pair of nonces with a server, then the server was, at some point prior, running the protocol with those nonces. We again find the necessary restrictions on the attacker to achieve this property. The property can only hold if the attacker does not acquire any of the secrets prior to the client agreeing on nonces. While one might expect that only the legitimacy of the signing key is necessary for authentication, if the attacker is able to obtain the PSK through

[2]We remind the reader that both the client and the server are equipped with long-term signing keys, and the corresponding public key certificates, for the purposes of authentication.

```
lemma secret_session_keys:
  "All tid actor peer write_key read_key peer_auth_status #i.
      SessionKey(tid, actor, peer, <peer_auth_status, 'auth'>, <write_key, read_key>)@i &
      not (Ex #r. RevLtk(peer)@r & #r < #i) &
      not (Ex tid3 x #r. RevDHExp(tid3, peer, x)@r & #r < #i) &
      not (Ex tid4 y #r. RevDHExp(tid4, actor, y)@r & #r < #i) &
      not (Ex resumption_master_secret #r. RevealPSK(actor, resumption_master_secret)@r) &
      not (Ex resumption_master_secret #r. RevealPSK(peer, resumption_master_secret)@r)
    ==> not Ex #j. K(read_key)@j"
```

Figure 5: secret_session_keys (Section 4.2.2)

```
lemma entity_authentication [use_induction, reuse]:
  "All tid actor peer nonces client_auth_status #i.
      CommitNonces(tid, actor, 'client', nonces)@i &
      CommitIdentity(tid, actor, 'client', peer, <client_auth_status, 'auth'>)@i &
      not (Ex #r. RevLtk(peer)@r & #r < #i) &
      not (Ex tid3 x #r. RevDHExp(tid3, peer, x)@r & #r < #i) &
      not (Ex tid4 y #r. RevDHExp(tid4, actor, y)@r & #r < #i) &
      not (Ex resumption_master_secret #r. RevealPSK(actor, resumption_master_secret)@r & #r < #i) &
      not (Ex resumption_master_secret #r. RevealPSK(peer, resumption_master_secret)@r & #r < #i)
    ==> (Ex tid2 #j. RunningNonces(tid2, peer, 'server', nonces)@j & #j < #i)"
```

Figure 6: entity_authentication (Section 4.2.3)

compromising cryptographic material, or the PSK directly, then the attacker is able to resume a session and impersonate the peer.

In addition to entity authentication, we consider a transcript agreement property, where the value agreed upon is a hash of the session transcript. This provides us with near-full agreement. However, there are a couple of notable omissions. Firstly, the protocol technically continues after the initial handshake, although none of these delayed handshake messages are included in the session transcript. Secondly, we observed that the actors do not necessarily agree on the current authentication status of the handshake, a situation we cover in more detail in Section 5.2.

Finally, we also prove an injective variant of mutual transcript agreement, which TLS naturally achieves by agreeing on fresh nonces. Hence, we show that TLS achieves a relatively strong authentication notion: mutual agreement on a significant portion of the state with recentness.

4.2.4 Uniqueness of the session keys. We prove in the straightforward way that for any two session keys generated, if they match then they must be from the same session. This holds without any restriction on the attacker, since it is a straightforward consequence of the actor generating a fresh nonce for each session. We do not prove anything about whether two session keys are related, since this trivially follows from the assumption of perfect cryptography.

4.2.5 Downgrade protection. The specification cites the work by Bhargavan et al. [11] for downgrade protection. This definition is not directly equivalent to any of Lowe's classical agreement methods; it only requires that both parties negotiate the *same* configuration parameters that they would do without the presence of an attacker. Specifically, we observe that agreeing on the parameters (in the sense of non-injective agreement) is sufficient to achieve this, but not necessary. Therefore, *within our model* we prove that TLS achieves downgrade protection through our authentication lemmas.

However, we note that this does not accurately capture the spirit of downgrade protection, due to the fact that we assume all cryptographic primitives are perfect and we do not model previous versions of TLS.

4.2.6 Forward secrecy with respect to long-term keys. The PFS property was briefly mentioned in the context of the long-term signing keys and the secrecy of session keys. However, in those cases, we did not cover the requirement for forward secrecy with regards to the PSK. We have an additional lemma `secret_session_keys_pfs` which captures that, in either a full DHE or PSK-DHE handshake, the secrecy of the session keys does not depend on the PSK remaining secret after the session is concluded.

To achieve this, we modify `secret_session_keys` as depicted in Figure 5, by adding a condition for the key-exchange mode, `not psk_ke_mode = psk_ke`, and loosening the restrictions on the attacker such that the `RevealPSK` action is only forbidden for any time point `#r < #i`. In proving this lemma, we show that the session keys are forward secure after a DHE.

4.2.7 Key Compromise Impersonation (KCI) resistance. Observant readers will notice that the only restriction on compromising long-term keys is that the peer's LTK must not be compromised. None of our security properties rely on the actor's LTK being hidden from the attacker[3]. Applying this fact to the authentication properties, therefore, additionally shows that the protocol, as given in the draft specification, achieves KCI resistance.

[3] A minor exception to this is that the attacker cannot use the actor's long-term key to impersonate the actor to themselves since in this case, the actor is also the peer.

4.3 Parameter Negotiation

The security of a TLS session critically depends on the integrity of the parameters negotiated during the corresponding TLS handshakes, in the initial and subsequent connections in the session. In TLS these parameters include the protocol version, the cipher suite, and the signature algorithm. Depending on the negotiated protocol version, additional values may or must be negotiated, such as the handshake mode, the DH group, and/or the PSK to be used.

For DH group negotiation, we model the client sending a list of two symbolic groups that the server can choose between. This feature allows us to provide a limited coverage of the HelloRetryRequest functionality of the protocol, which we address in more detail in Section 3.4. We also provide support for PSKs but limit the number of PSKs offered to one per handshake.

By using Tamarin, we prove that the client and the server agree on the transcript of the protocol, and thus on the values selected during negotiation. This means that an attacker cannot force the client or the server to accept a value that they did not initially offer.

There are two main classes of parameter negotiation attacks: forcing the use of bad cipher suites [1, 16] or bad signature algorithms [9], and forcing the use of older and insecure versions of SSL/TLS [5, 44]. Because we model perfect cryptography and cover TLS 1.3 only, we consider these attacks to be out of scope (cf. Section 4.2.5).

5 ANALYSIS AND RESULTS

In this section we provide a detailed description of our analysis, including a discussion of our results and an exploration of an authentication anomaly uncovered by our work.

In general we find that TLS 1.3 meets the properties outlined in the specification that our modelling process was able to capture. We show that TLS 1.3 enables a client and a server to agree on secret session keys and that these session keys are unique across, as well as within, handshake instances. Our analysis shows that PFS of session keys holds in the expected situations, i.e., in the (EC)DHE and PSK+(EC)DHE handshake modes. We also show that TLS 1.3, by and large, provides the desired authentication guarantees in both the unilateral and mutual authentication cases. The situation in which this is not the case is covered in the section to follow.

We remind the reader that our model does not truly cover downgrade protection, or the protection of endpoint identities at this time. A treatment of downgrade protection across TLS protocol versions would require modelling the earlier versions of TLS in a way that is consistent with the TLS 1.3 model as developed here. To consider the downgrade protection of cipher suites, we would need to relax our current assumption of perfect cryptography through rules that, for instance, allow for an attacker to learn the payload of a particular kind of encrypted messages without knowing the key. In spite of the fact that these additional considerations would substantially complicate the model and the proof process, our model is perfectly suited to their inclusion and could form the basis of future work.

5.1 Positive results

We now present our results for TLS 1.3, commenting on our proof methods and findings.

5.1.1 Proof strategies. For models as complex as TLS 1.3, proving lemmas in Tamarin is a multi-stage process, and proving complex lemmas directly is often infeasible. For protocol models of this size the proof trees can become very large. Tamarin provides a number of features that allow complex proofs to be broken down into more manageable sections. Writing sublemmas provides hints to the Tamarin constraint solving algorithm, allowing it to solve complex sections of a larger proof directly, making the overall proof more manageable. For the TLS 1.3 model, we used several types of lemmas. Helper lemmas can be used to quickly solve repetitive sections of a larger proof without repeatedly unrolling the entire subtree. Typing lemmas provide hints to the Tamarin engine about the potential sources of messages, reducing the branching of a proof tree. Inductive lemmas instruct Tamarin to prove the lemmas inductively, allowing us to break out of loops in the protocol, which otherwise can produce infinite proof trees. Proving the main properties of TLS 1.3 required many helper lemmas, of all of these types.

The Tamarin engine can also use heuristics to auto-prove lemmas, which proved invaluable in quickly re-proving large sections of properties after making changes to the model. By investing time in writing auto-provable sublemmas, we could flexibly incorporate changes made to the specification without having to restart our analysis from scratch.

The more complex lemmas used in our analysis of TLS 1.3, however, required manual proving in the Tamarin interactive prover. We note that by manual proving in this context we mean manually guiding the Tamarin prover through a proof by using the Tamarin graphical user interface.

Using the `m4` preprocessor to generate restricted subsets of the model we were able to prototype lemmas in a simpler environment without expending unnecessary effort. To give an indication of the number of helper lemmas required, and the relationship between all of our lemmas, we have constructed a 'lemma map', displayed in Figure 7. The map also indicates which lemmas were auto-proved by Tamarin, and which ones needed manual guidance for Tamarin to prove them.

In total, the modelling effort represents approximately 3 months worth of work. However, the vast majority of that is the process of writing lemmas to break down the overall proving effort into smaller, autoprovable chunks. With these lemmas in place, proving the entire model takes about a week of work, and significant computing resources. The model itself takes over 10GB RAM just to load, and can easily consume 100GB RAM in the course of a proof. In one instance, an automatically-computed proof was almost 1 million lines long. Once the proofs have been produced, they can be verified in the space of about a day, although still requiring a vast amount of RAM.

5.1.2 Findings. We summarise our results in Table 1. For each property discussed in Sections 2 and 4, we indicate our findings. We use $*$ to indicate that the property holds in most situations. Cases in which the property does not hold to the expected degree, are covered in sections to follow. We also list the applicable Tamarin lemma(s).

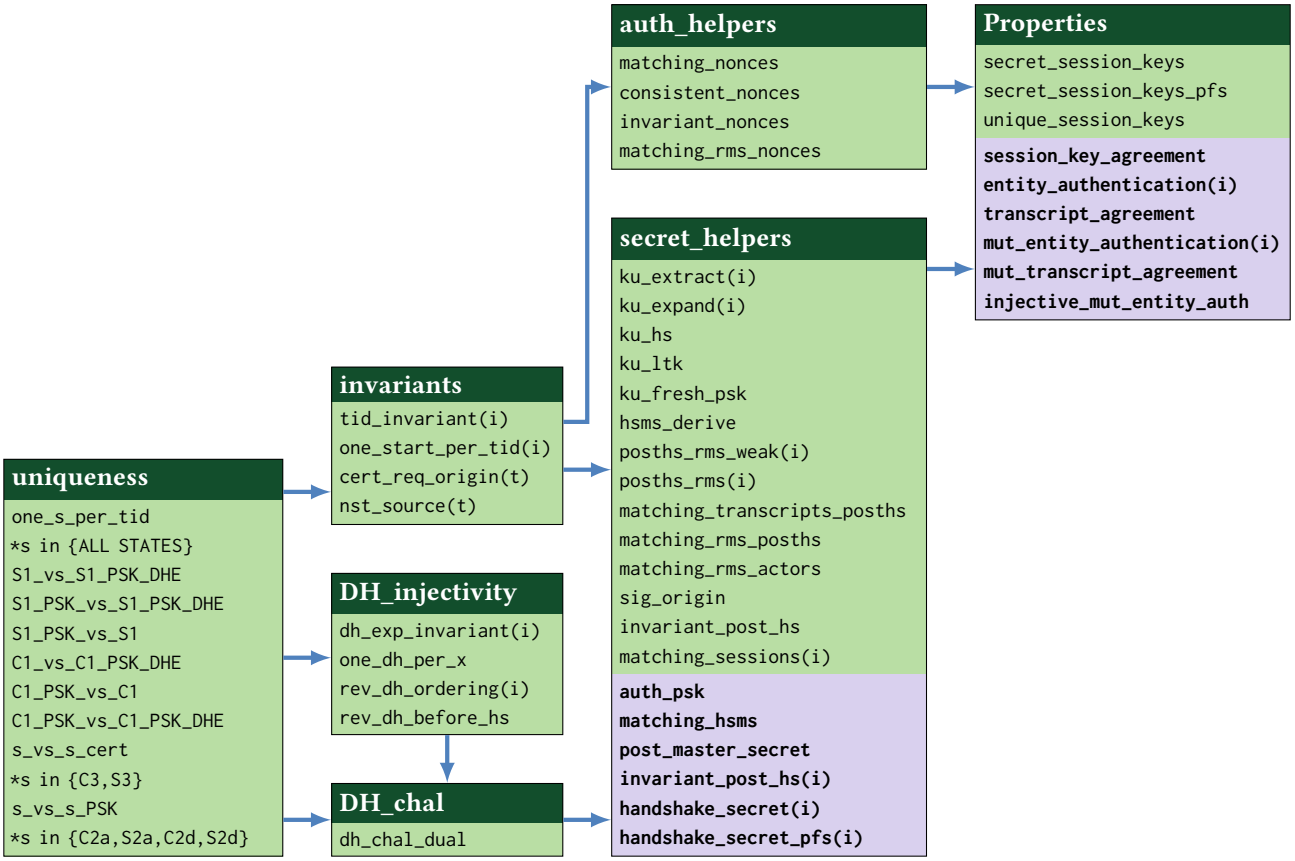

Figure 7: Lemma Map. Bold lemma names with a purple background indicate where manual interaction via the Tamarin visual interface was required. The remaining lemmas were automatically proven by Tamarin, without manual interaction. An arrow from one category to another implies that the proof of the latter depends on the former. The Properties box contains the main TLS 1.3 properties.

Property proven	Lemma(s)
(1) Same session keys	`session_key_agreement`
(2) Secret session keys	`secret_session_keys`
(3) Peer authentication*	`entity_authentication` `mutual_entity_authentication`
(4) Unique session keys	`unique_session_keys`
(6) Perfect forward secrecy	`secret_session_keys_pfs`
(7) Key compromise impersonation	`entity_authentication` `mutual_entity_authentication`

Table 1: TLS 1.3 Tamarin results

5.2 Possible mismatch between client and server view

During the development of our model, and in particular the analysis of the post-handshake client authentication, we encountered a possible behaviour that suggested that TLS 1.3 fails to meet certain strong authentication guarantees.

While there are many definitions of authentication, the common thread among strong authentication guarantees is that both parties share a common view of the session, i.e. that they agree on exchanged data, keys, etc. During our analysis of the post-handshake client authentication, it became apparent that the client does not receive any explicit confirmation that the server has successfully received the client's response. Due to the asynchronous nature of the post-handshake client authentication, the client may keep receiving data from the server, and will not be able to determine if the server has received its authentication message. As a consequence, the client cannot be sure whether the server sent the data under the assumption that the client is authenticated.

We formally modelled this property by adding a variable to the client and the server that records the current status of the connection, and in particular, if the connection is unilaterally or mutually authenticated. We discovered that even when the server asks for a post-handshake client authentication, and the client responds, the client cannot be sure that the server considers the channel to be mutually authenticated.

In concurrent work by Bhargavan et al. [10], a similar issue was uncovered for the 0.5-round trip time (RTT) case. A discussion with the TLS 1.3 working group revealed that an equivalent problem also exists within the main handshake. During the main handshake, the

server can request a client certificate, and may decide to reject the certificate (for example because it violates certain domain-specific policies), *but still continue with the connection* as if the certificate were accepted. Therefore, the client cannot be sure (after what appears to be a main handshake with mutual authentication) that the server considers the client to be authenticated. Thus, this phenomenon leaves the client in the dark about whether or not the server considers it to be authenticated, even though the server asked for a certificate and the client supplied it.

To see why this may become a problem at the application level, consider the following application. Imagine a client and a server that implement TLS 1.3, where the server has the following policy: any data received over a mutually authenticated connection are stored in a secure database; all data received over connections where the client is not authenticated are stored in an insecure log. The client connects, the server requests a certificate, which the client duly provides, but the server rejects and continues regardless. Since the server rejected the certificate, it continues to store incoming messages in the insecure log. However, the client may assume it has been authenticated, and start sending sensitive data, which ends up in the insecure log.

The TLS working group has decided not to fix this behaviour for TLS 1.3, and has not introduced any mechanism that informs the client of the server's view of the client's authentication status. If a client wants to be sure that the server considers it to be authenticated, this needs to be dealt with at the application layer. We anticipate that some client applications will incorrectly assume that sending a client certificate and obtaining further server messages indeed guarantees that the server considers the connection to be mutually authenticated. As we have shown, this is not the case in general, and may lead to serious security issues despite there being no direct violation of the specified TLS 1.3 security requirements.

6 THE RELATION BETWEEN OUR MODEL AND THE TLS 1.3 SPECIFICATION

While there have been many academic analyses of various drafts of TLS 1.3 [4, 10, 22, 24, 25, 28, 30, 33, 35, 38], they all (explicitly or implicitly) consider only part of the specification. Most analyses, even those that claim to be "complete" do not consider all possible modes, and many manual cryptographic analyses consider modes only in isolation (and not their interaction). This is caused by the inherent complexity of analysing TLS 1.3 and is not a problem in itself; rather, it justifies the need for multiple approaches.

However, we are of the opinion that readers, regardless of whether or not they are experts in the field, should be able to easily deduce the exact coverage of a given analysis. To ensure this, we provide an unprecedented level of transparency concerning the relationship between our model and the RFC (the draft specification) by creating a website [21] that contains an annotated version of the RFC. Consider the following excerpt:

In the above excerpt, the left-hand side is a direct copy from the RFC, and the right-hand side contains our annotations. For example, they show how the concrete data structures of TLS 1.3 are mapped into abstract term structures. Additionally, we annotate the prose, describing the possible behaviours so as to indicate which Tamarin rules model them. The annotations also show exactly which details we do not model (and often list the reasons why).

We used these annotations ourselves during the development of our model to keep track of the parts of the specification that we had already modelled, and how we modelled them, which also simplified the task of keeping track of updates to the specification, something which proved incredibly useful given the rapid pace at which the draft specification would undergo changes.

Our annotated RFC has a number of desirable features:

- Readers can check which parts we abstracted, and how, without having to reinvent the mapping between the Tamarin model and the RFC themselves. In other words, one can read through our website to see what is covered, and how it is covered, without having to understand Tamarin's formalism.
- If the specification is updated or changed, we can immediately track where the model should be changed.

We encourage other analyses of TLS 1.3 to follow a similar transparent approach, which would help the community to better understand which details from the specification might still need to be covered. We envision this will enable a faster convergence of confidence in all the details of the standard.

7 CONCLUSIONS

In this work we modelled the current draft of the TLS 1.3 specification within the symbolic analysis framework of the Tamarin prover, and used the tool to verify the majority of the security guarantees that TLS 1.3 claims to offer its users.

We focus on ruling out complex interaction attacks by considering an unbounded number of concurrent connections, and all of the TLS 1.3 handshake modes. We cover both unilateral and mutual authentication, as well as session key secrecy in all of the TLS 1.3 handshake modes with respect to a Dolev-Yao attacker. We also capture more advanced security properties such as perfect forward secrecy and key compromise impersonation. Our Tamarin model covers substantially more interactions than previous analyses due to its modularity.

Besides verifying that draft 21 of the TLS 1.3 specification meets the claimed security properties in most of the handshake modes and variants, we also discover an unexpected authentication behaviour which may have serious security implications for implementations of TLS 1.3. This unexpected behaviour, at a high level, implies that TLS 1.3 provides no direct means for a client to determine its authentication status from the perspective of a given server. As a server may treat authenticated data differently to unauthenticated data, the client may end up in position in which its sensitive data gets processed as non-sensitive data by the server.

During the course of our analysis we also developed a line-by-line modelling aide that accurately captured which parts of the specification we were able to model, and which parts were abstracted. This artifact allows us to easily assess the faithfulness and coverage of our model, and also makes our model highly amenable to all kinds of extensions, especially with respect to the security properties and threat model. We expect that this artifact may serve as a comprehensive informational aide to academic researchers and well as the TLS Working Group.

```
Certificate Verify

This message is used to provide explicit proof that an endpoint possesses the
private key corresponding to its certificate and also provides integrity for
the handshake up to this point. Servers MUST send this message when
authenticating via a certificate. Clients MUST send this message whenever
authenticating via a Certificate (i.e., when the Certificate message is
non-empty). When sent, this message MUST appear immediately after the
Certificate message and immediately prior to the Finished message.

Structure of this message:

%%% Authentication Messages

   struct {
       SignatureScheme algorithm;
       opaque signature<0..2^16-1>;
   } CertificateVerify;
The algorithm field specifies the signature algorithm used (see
Section 4.2.3 for the definition of this field).  The signature is a
digital signature using that algorithm.  The content that is covered
under the signature is the hash output as described in Section 4.4,
namely:

   Transcript-Hash(Handshake Context, Certificate)
---snip---
```

```
We compute the (server) signature as:

messages = <messages, Certificate>
signature = compute_signature(~ltkS, server)
where compute_signature expands to:

sign{<'TLS13_server_CertificateVerify', h(messages)>}
Since messages contains the handshake transcript up until that point, this is
valid for Handshake Context. We do not attempt to add the padding prefix
specified in the specification since it would have no purpose given our
assumption of perfect crypto.

The CertificateVerify message is simply defined as:

define(<!CertificateVerify!>, <!handshake_record('15', $sig_alg, signature)!>)
We do not currently model using different signing algorithms or their effects
on security.

The peer validates the CertificateVerify message by recomputing the signature
input, and enforcing the action Eq(verify(signature, sig_messages, pk(~ltkS)),
true) which makes the trace invalid if the verification fails (implying the
peer terminates the connection if receiving an invalid signature).

Note that an alternative way to model this in Tamarin would be to provide the
peer with the long-term key ~ltkA and pattern match the signature as an
expected message. While this can (probably) be shown to be equivalent and is
potentially more efficient for Tamarin, we believe using explicit verification
is clearer.
```

Figure 8: An excerpt of our website, showing how we annotated the specification. The full version can be found at [21].

ACKNOWLEDGMENTS

This work was supported by the Engineering and Physical Sciences Research Council, grant number EP/K035584/1, the European Research Council, grant number 610150, and the Air Force Office of Scientific Research, grant number FA9550-17-1-0206.

REFERENCES

[1] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. 2015. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. (2015), 13 pages. https://doi.org/10.1145/2810103.2813707

[2] Nadhem J. AlFardan and Kenneth G. Paterson. 2012. Plaintext-Recovery Attacks Against Datagram TLS. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. http://www.internetsociety.org/plain-text-recovery-attacks-against-datagram-tls

[3] Nadhem J. AlFardan and Kenneth G. Paterson. 2013. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. 526–540. http://dx.doi.org/10.1109/SP.2013.42

[4] Kenichi Arai and Shin'ichiro Matsuo. 2016. Formal Verification of TLS 1.3 Full Handshake Protocol Using ProVerif. TLS mailing list post. (February 2016). https://www.ietf.org/mail-archive/web/tls/current/msg19339.html

[5] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. 2016. DROWN: Breaking TLS with SSLv2. In *5th USENIX Security Symposium*. 689–706.

[6] Gregory V. Bard. 2004. The Vulnerability of SSL to Chosen Plaintext Attack. *IACR Cryptology ePrint Archive* 2004 (2004), 111. http://eprint.iacr.org/2004/111

[7] Gregory V. Bard. 2006. A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL. In *SECRYPT 2006, Proceedings of the International Conference on Security and Cryptography, Setúbal, Portugal, August 7-10, 2006, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*. 99–109.

[8] Mihir Bellare and Phillip Rogaway. 1993. Entity authentication and key distribution. In *Annual International Cryptology Conference*. Springer, 232–249.

[9] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. 2015. A Messy State of the Union: Taming the Composite State Machines of TLS. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 535–552. http://dx.doi.org/10.1109/SP.2015.39

[10] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. *Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate*. Technical Report. INRIA.

[11] Karthikeyan Bhargavan, Christina Brzuska, Cédric Fournet, Matthew Green, Markulf Kohlweiss, and Santiago Zanella-Béguelin. 2016. Downgrade resilience in key-exchange protocols. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 506–525.

[12] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jianyang Pan, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, and Jean Karim Zinzindohoué. 2016. *Implementing and Proving the TLS 1.3 Record Layer*. Technical Report. INRIA. http://eprint.iacr.org/2016/1178.

[13] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. 2014. Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. 98–113. http://dx.doi.org/10.1109/SP.2014.14

[14] K. Bhargavan, N. Kobeissi, and B. Blanchet. 2016. ProScript TLS: Building a TLS 1.3 Implementation with a Verifiable Protocol Model. (2016). Presented at TRON 1.0, San Diego, CA, USA, February 21.

[15] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 456–467. https://doi.org/10.1145/2976749.2978423

[16] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In *Network and Distributed System Security Symposium–NDSS 2016*.

[17] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. 2016. Proverif 1.96: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial List of Figures. (2016). http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf

[18] Daniel Bleichenbacher. 1998. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*. 1–12. http://dx.doi.org/10.1007/BFb0055716

[19] Ran Canetti and Hugo Krawczyk. 2001. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*. 453–474. http://dx.doi.org/10.1007/3-540-44987-6_28

[20] Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. 2003. Password Interception in a SSL/TLS Channel. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. 583–599. http://dx.doi.org/10.1007/978-3-540-45146-4_34

[21] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. Source files and annotated RFC for TLS 1.3 analysis. (2017). https://tls13tamarin.github.io/TLS13Tamarin/.

[22] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. 2016. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE,

470–485.

[23] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.

[24] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2015. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*. 1197–1210. http://doi.acm.org/10.1145/2810103.2813653

[25] 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. (2016). http://eprint.iacr.org/.

[26] Thai Duong and Juliano Rizzo. 2011. Here Come the ⊕ Ninjas. Unpublished manuscript. (May 2011).

[27] Thai Duong and Juliano Rizzo. 2012. The CRIME Attack. Ekoparty Security Conference presentation. (2012).

[28] Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. 2016. Key Confirmation in Key Exchange: A Formal Treatment and Implications for TLS 1.3. In *2016 IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 23-25, 2016*.

[29] Christina Garman, Kenneth G Paterson, and Thyla Van der Merwe. 2015. Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS.. In *USENIX Security*. 113–128.

[30] Marko Horvat. 2016. *Formal Analysis of Modern Security Protocols in Current Standards*. Ph.D. Dissertation. University of Oxford.

[31] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. 2015. On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*. 1185–1196. http://doi.acm.org/10.1145/2810103.2813657

[32] Vlastimil Klíma, Ondrej Pokorný, and Tomás Rosa. 2003. Attacking RSA-Based Sessions in SSL/TLS. In *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*. 426–440. http://dx.doi.org/10.1007/978-3-540-45238-6_33

[33] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. 2014. (De-)Constructing TLS. *IACR Cryptology ePrint Archive* 2014 (2014), 20. http://eprint.iacr.org/2014/020

[34] Hugo Krawczyk. 2010. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*. 631–648. http://dx.doi.org/10.1007/978-3-642-14623-7_34

[35] Hugo Krawczyk and Hoeteck Wee. 2015. The OPTLS Protocol and TLS 1.3. *IACR Cryptology ePrint Archive* 2015 (2015), 978. http://eprint.iacr.org/2015/978

[36] Brian A. LaMacchia, Kristin E. Lauter, and Anton Mityagin. 2007. Stronger Security of Authenticated Key Exchange. In *Provable Security, First International Conference, ProvSec 2007, Wollongong, Australia, November 1-2, 2007, Proceedings*. 1–16. http://dx.doi.org/10.1007/978-3-540-75670-5_1

[37] A. Langley and W. Chang. 2013. QUIC Crypto. (June 2013). Available at https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/.

[38] Yong Li, Sven Schäge, Zheng Yang, Florian Kohlar, and Jörg Schwenk. 2014. On the Security of the Pre-shared Key Ciphersuites of TLS. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*. 669–684. http://dx.doi.org/10.1007/978-3-642-54631-0_38

[39] Gavin Lowe. 1997. A Hierarchy of Authentication Specifications. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations (CSFW '97)*. IEEE Computer Society, Washington, DC, USA, 31–. http://dl.acm.org/citation.cfm?id=794197.795075

[40] Colm MacCárthaigh. 2017. Security review of TLS1.3 0-RTT. TLS mailing list post. (May 2017). https://www.ietf.org/mail-archive/web/tls/current/msg23051.html Available at https://www.ietf.org/mail-archive/web/tls/current/msg23051.html.

[41] Itsik Mantin. 2015. Attacking SSL when using RC4. White Paper. (March 2015). https://www.imperva.com/docs/HII_Attacking_SSL_when_using_RC4.pdf

[42] Nikos Mavrogiannopoulos, Frederik Vercauteren, Vesselin Velichkov, and Bart Preneel. 2012. A cross-protocol attack on the TLS protocol. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. 62–72. http://doi.acm.org/10.1145/2382196.2382206

[43] Bodo Moeller. 2004. Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures. Unpublished manuscript. (May 2004). http://www.openssl.org/~bodo/tls-cbc.txt.

[44] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. 2014. This POODLE bites: Exploiting The SSL 3.0 Fallback. Security Advisory. (September 2014). https://www.openssl.org/~bodo/ssl-poodle.pdf

[45] Kenneth G. Paterson and Thyla van der Merwe. 2016. Reactive and Proactive Standardisation of TLS. In *Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings*. 160–186.

[46] E. Rescorla. 2015. The Transport Layer Security (TLS) Protocol Version 1.3 (draft, revision 10). (October 2015). https://tools.ietf.org/html/draft-ietf-tls-tls13-10 Available at https://tools.ietf.org/html/draft-ietf-tls-tls13-10.

[47] Eric Rescorla and Brian Korver. 2003. Guidelines for writing RFC text on security considerations. RFC 3552 (Informational). (July 2003). https://tools.ietf.org/html/rfc3552

[48] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. 2012. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, Stephen Chong (Ed.). IEEE, 78–94.

[49] Serge Vaudenay. 2002. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS .... In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*. 534–546. http://dx.doi.org/10.1007/3-540-46035-7_35

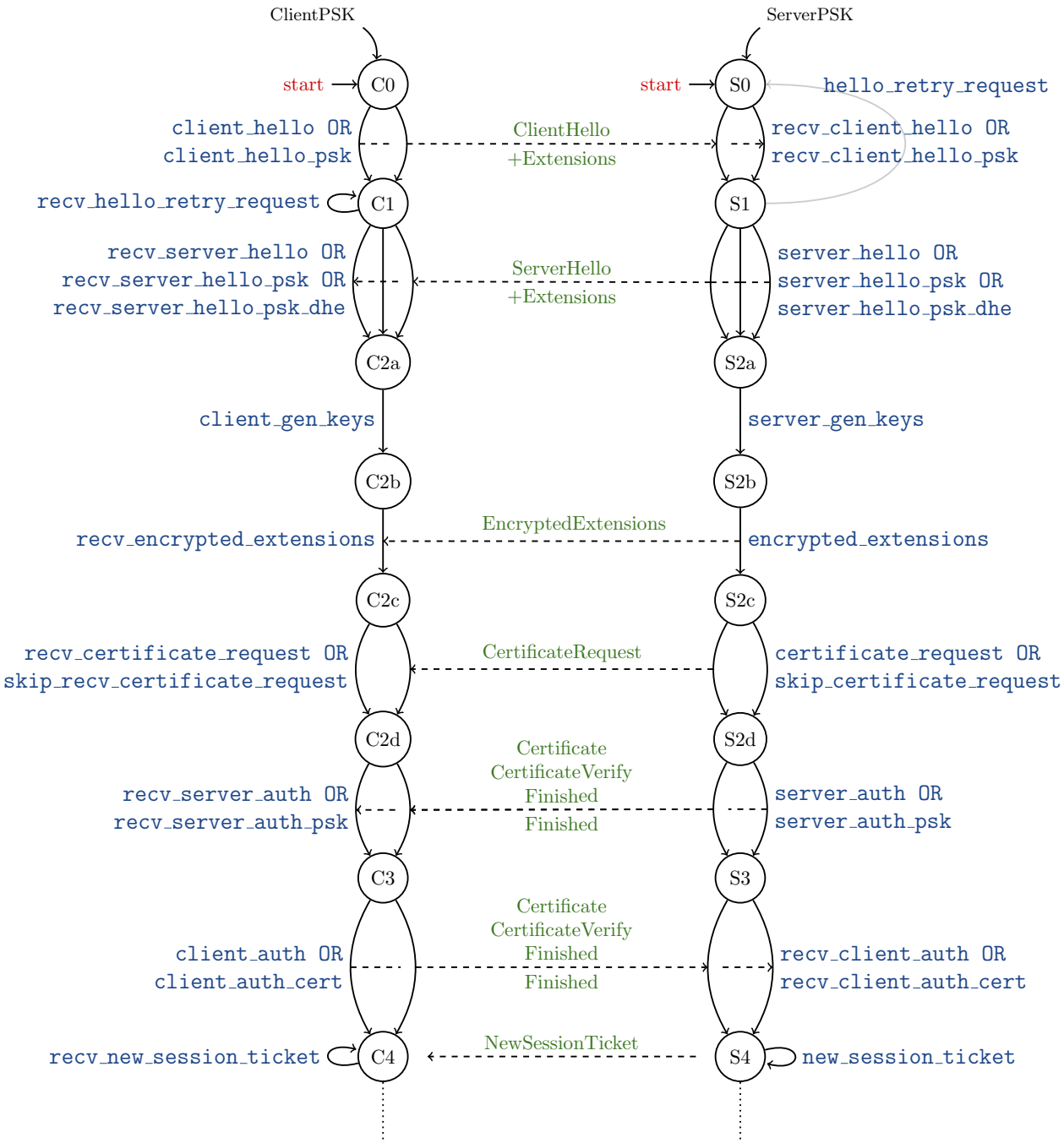

Figure 9: Part 1 of the full state diagram for Tamarin model, showing all rules covered in the initial handshake (excluding rules dealing with record layer).

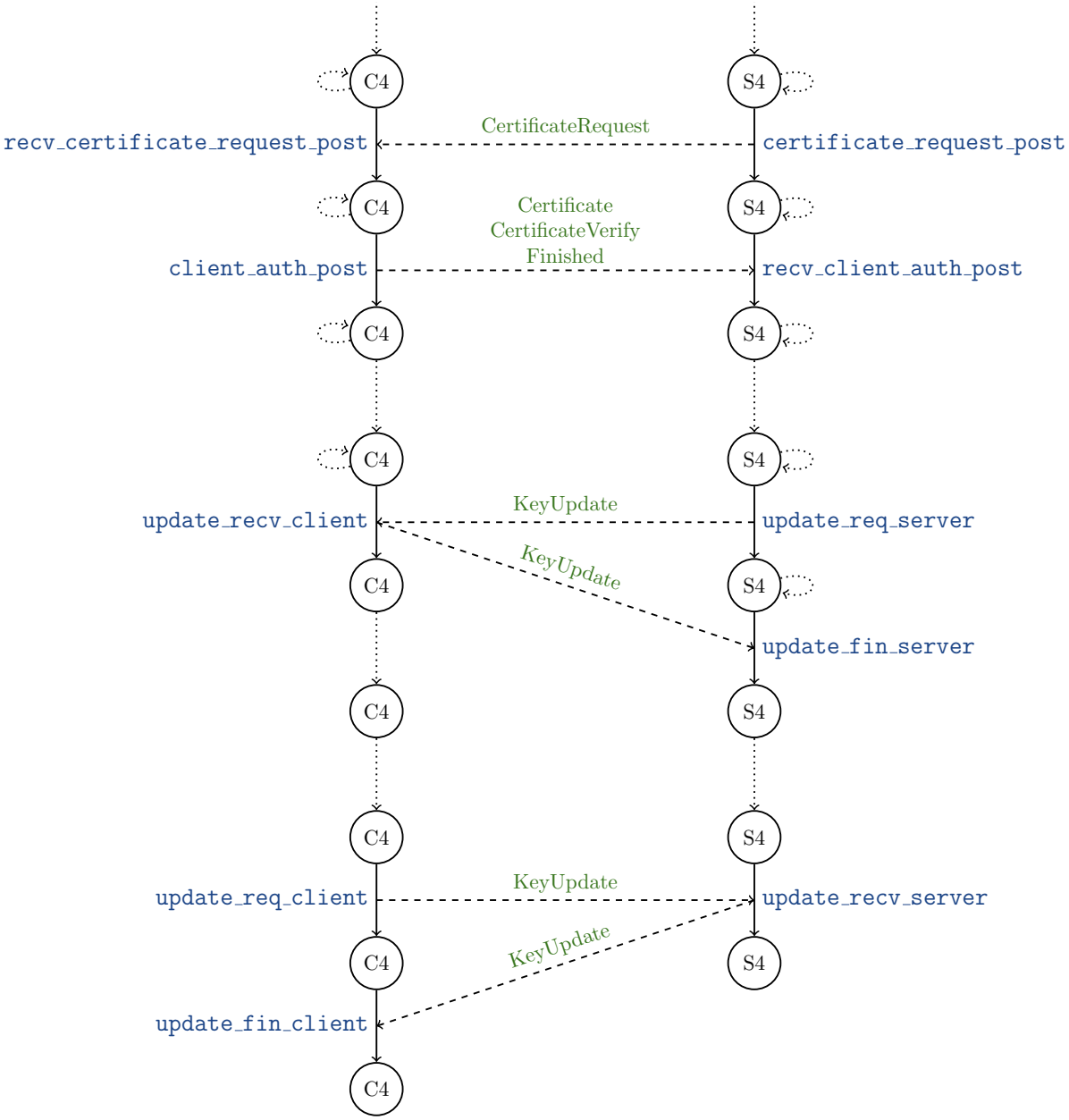

Figure 10: Part 2 of the full state diagram for Tamarin model, showing all post-handshake rules covered.