Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication

(Preprint, February 10, 2016)

Cas Cremers, Marko Horvat
Department of Computer Science
University of Oxford, UK

Sam Scott, Thyla van der Merwe
Information Security Group
Royal Holloway, University of London, UK

Abstract—**After a development process of many months, the TLS 1.3 specification is nearly complete. To prevent past mistakes, this crucial security protocol must be thoroughly scrutinised prior to deployment.**

In this work we model and analyse the latest draft of the TLS 1.3 specification, namely revision 10, using the Tamarin prover, a tool for the automated analysis of security protocols. We specify and analyse the interaction of various handshake modes for an unbounded number of concurrent TLS connections. We show that revision 10 meets the goals of authenticated key exchange in both the unilateral and mutual authentication cases.

We extend our model to incorporate the desired delayed client authentication mechanism, a feature that is likely to be included in the next revision of the specification, and uncover a potential attack in which an adversary is able to successfully impersonate a client during a PSK-resumption handshake. This observation was reported to, and confirmed by, the IETF TLS Working Group.

Our work not only provides the first supporting evidence for the security of several complex protocol mode interactions in TLS 1.3, but also shows the strict necessity of recent suggestions to include more information in the protocol's signature contents.

1. Introduction

The TLS protocol is used globally by millions of users on a daily basis, serving as the core building block for Internet security. However, TLS is also a very complex protocol with many possible variants and use cases, which has complicated thorough cryptographic analysis for decades. Although TLS has received much attention since its deployment by Netscape as SSL in 1995, it is not since the double menace of BEAST [19] in 2011 and CRIME [20] in 2012 that the protocol has become the subject of intense analysis and academic study; prior to the release of these attacks, we see a number of relevant works spanning almost two decades [6], [7], [15], [16], [28], [29], [31], [42], [43], [44], [53], [54]. Post-2011, we see a comparable number of works in less than 5 years [2], [3], [4], [5], [10], [11], [12], [13], [14], [18], [23], [26], [27], [30], [33], [34], [36], [39], many representing great advances on both the manual and automated fronts and resulting in the discovery of many weaknesses.

The various flaws identified in TLS 1.2 [17] and below, be they implementation- or specification-based, have prompted the TLS Working Group to adopt an 'analysis-before-deployment' design paradigm in drafting the next version of the protocol, TLS 1.3 [47]. Most notably, the cryptographic core of the new TLS handshake protocol is largely influenced by the OPTLS protocol of Krawczyk and Wee [34], a protocol that has been expressly designed to offer zero Round-Trip Time (0-RTT) exchanges and ensure perfect forward secrecy. Its simple structure lends itself to analysis via manual and automated means, a benefit that was deemed desirable for TLS 1.3. Although the logic of the protocol has been simplified, the addition of 0-RTT functionality as well as the new resumption and client authentication mechanisms has introduced new complexity.

The overall complexity of TLS 1.3 implies that to perform a truly complete cryptographic analysis (either manual or tool-supported) of the entire protocol would be a substantial undertaking, and unlikely to be completed in time for the release of TLS 1.3.

However, given the critical importance of TLS, it is paramount that the design of the TLS 1.3 protocol is critically analysed to minimise the number of potential flaws before the protocol is finalised and deployed. Our work based on tool-supported, symbolic verification of the TLS 1.3 security guarantees contributes towards this goal.

1.1. Contributions

Our main contribution is a more comprehensive treatment of the TLS 1.3 specification than previous works [18], [30], [34]. Additionally, our formal model of TLS 1.3 serves as a tool that can be extended and modified for future releases, and therefore results in a longer-lasting benefit to the designers of the TLS protocol. We describe our contributions in more detail below.

Comprehensive analysis. One of the most relevant and up-to-date analyses pertaining to TLS 1.3 is arguably the analysis of OPTLS by its designers [34]. The authors note that their analysis is not intended to cover the full TLS 1.3 specification. In particular, they only consider the different handshake modes in isolation. Also, they include neither client authentication nor resumption It is precisely this gap that we aim to fill with our work.

We formally model and analyse TLS 1.3 revision 10 (henceforth referred to as `draft-10`[1]). With TLS 1.3 due to be released in early 2016, it is expected that `draft-10` will closely resemble the final version. With the exception of the client authentication modifications, it is unlikely that any future changes will dramatically alter our analysis. However, we will ensure that any future changes are incorporated into our model.

Our analysis complements the work from [34] and previous works [18], [30] by covering the following aspects:

- The security of, and secure interaction of, the following handshake modes: regular (EC)DHE mode, (Pre-Shared Key) PSK mode, PSK-DHE mode and 0-RTT mode.
- The PSK-resumption handshake when composed with any acceptable initial handshake, namely, an (EC)DHE handshake, a PSK handshake, a PSK-DHE handshake and a 0-RTT handshake.
- The security of the proposed delayed authentication mechanism in the context of all previous modes.
- A near-complete coverage of the state transitions in the standard. Previous works have abstracted away message components or subprotocols, or were developed before the newer mechanisms have crystallised.

For our analysis, we use the Tamarin prover [50], a state-of-the-art tool for the symbolic analysis of security protocols. The Tamarin framework enables us to precisely specify and analyse the secrecy and complex authentication properties of the various handshake modes. Furthermore, Tamarin's multiset rewriting semantics are well-suited for modelling the complex transition system implied by the TLS 1.3 specification; the tool allows for analysing the interaction of the assorted handshake modes as well as an unbounded number of concurrent TLS sessions.

We consider a Dolev-Yao adversary model in which the adversary can also reveal long-term private keys of honest parties. Our Tamarin model includes both the client authentication mechanism and session resumption, so our property specifications go well beyond the basic session key secrecy considered in [34].

We find that `draft-10` achieves the standard goals of authenticated key exchange. In particular, we show that a client has assurances regarding the secrecy of the established session key, as well as assurances regarding the identity of the server with whom it has established this key. The server obtains equivalent assurances when authenticating the client in both the standard way, and when using the newly introduced 0-RTT mechanism. Our analysis confirms perfect forward secrecy of session keys and also covers the properties of handshake integrity and secrecy of early data keys. We verify these desirable properties in the presence of composable handshake modes and an unbounded number of TLS sessions, something which has not been done in previous TLS 1.3 analyses.

The discussion arising from the TLS Working Group suggests a new delayed authentication mode is likely to appear in future revisions of the TLS 1.3 specification [45]. Our exploration of this option has resulted in the discovery of a potential attack. Specifically, an adversary is able to impersonate a client when communicating with a server owing to a vulnerability in the client authentication mechanism of the PSK-resumption handshake. Our attack highlights the strict necessity of creating a channel binding between TLS 1.3 handshakes.

.

Future prospects. The scope of our model and analysis goes well beyond `draft-10`. As the final modifications are made to the TLS 1.3 specifications, the model will be updated and the analysis re-run. This will ensure that no new errors are introduced with respect to current properties, and will substantially simplify the analysis of any new properties that may arise. Thus, we expect that our analysis will help to inform and guide the final stages of the TLS 1.3 design.

Acknowledgements. We would like to thank Eric Rescorla and Martin Thomson of Mozilla and the TLS Working Group for their invaluable inputs to this work by way of numerous clarifying conversations.

1.2. Related work on TLS 1.3

The 0-RTT mechanism of OPTLS, and hence of TLS 1.3, is similar to that of Google's Quick UDP Internet Connections (QUIC) protocol [35]. Lychev et al. introduce a security model for what they term *Quick Connections (QC)* protocols and analyse QUIC within this framework [38]. Although they do not focus on TLS 1.3, they do point out that the 0-RTT mode of TLS fits the definition of a QC protocol. Fischlin and Günther also provide an analysis of QUIC [22] by developing a Bellare-Rogaway style model for *multi-stage* key exchange protocols.

Both QUIC and the TLS 1.3 handshake protocol can be viewed as multi-stage key exchange protocols because the communicating parties establish multiple session keys during an exchange, potentially using one key to derive another. Fischlin and Günther show QUIC to be secure within this model and in work by Dowling et al. [18], two TLS 1.3 drafts, specifically `draft-05` and `draft-dh` are analysed using this framework. Although the authors showed that keys output by the handshake protocol could be securely used by the record protocol, the timing of their analysis was unfortunate; at the time of writing, the TLS drafts did not include a 0-RTT mode and resumption had not yet been merged with the PSK mode. Kohlweiss et al. also produced an analysis of `draft-05` using a constructive-cryptography approach [30].

[1] We borrow this naming convention from [18].

Although there were changes including a reduction in handshake latency, removal of renegotiation and a switch to AEAD ciphers in the earlier drafts of TLS 1.3, it is not until `draft-07` that we see a radical shift in the design of the protocol away from TLS 1.2. Hence, we argue that the results described above may not easily transfer to later drafts. From `draft-07` onwards, we see the adoption of the OPTLS protocol of Krawczyk and Wee [34] as the foundation for TLS 1.3. Not only is there the inclusion of 0-RTT support and a switch to a semi-ephemeral Diffie-Hellman exchange as is the case in OPTLS, but also the new resumption mechanism that makes use of PSKs.

1.3. Paper organisation

In Section 2 we introduce the main new features of the TLS 1.3 protocol and its stated security goals. We describe how we formally model the protocol and its complex set of behaviours in Section 3. We proceed in Section 4 by formally specifying a range of secrecy and authentication properties that apply to different use cases, and analyse the protocol with respect to these properties. We consider the addition of client authentication in PSK mode in Section 5. We conclude in Section 6, where we also discuss future work.

2. TLS 1.3: New mechanisms, stated goals, and security properties

We introduce the new mechanisms of TLS 1.3 in comparison to TLS 1.2. We then present the protocol's intended security properties as described by the specification.

2.1. Design

The main design goals for TLS 1.3 include [52]:

- encrypt as much of the handshake as possible,
- re-evaluate the handshake contents,
- reduce handshake latency—one Round-Trip Time (1-RTT) for full handshakes, zero Round-Trip Time (0-RTT) for repeated handshakes, and
- update the record protection mechanisms.

We now discuss how TLS 1.3 implements these four requirements, as well as its key derivation procedures.

Handshake encryption. The motivation behind handshake encryption is to reduce the amount of observable data to both passive and active adversaries [52]. In contrast to TLS 1.2, which only provides communicating entities with session keys to protect application data, TLS 1.3 provides for the establishment of additional session keys to be used for resumption and handshake encryption purposes. Handshake encryption begins immediately after the handshake keys have been negotiated via a Diffie-Hellman (DH) exchange.

Handshake contents. As will be discussed in the following section, the handshake structure has been reworked for efficiency purposes. An additional server message has been added to accommodate the event of a parameter mismatch, and compression has been removed. Static DH and RSA have been removed in favour of the PFS-supporting finite field ephemeral Diffie- Hellman (DHE) and elliptic-curve ephemeral Diffie-Hellman (ECDHE) key exchange modes. RSA certificates are still being used for the transcript-signing keys in both the DHE and ECDHE modes (alongside ECDSA certificates). Server-side signatures have been mandated in all handshake modes.

Handshake latency. The TLS 1.2 handshake required a two Round-Trip Time exchange prior to communicating entities being able to transmit application data. The handshake has been reworked in TLS 1.3 to require just 1-RTT if no parameter mismatches occur.

TLS 1.3 also includes a 0-RTT option in which the client is able to send application data as part of its first flight of messages, offering a clear efficiency advantage over TLS 1.2. This functionality is enabled by a server providing a long-term (EC)DH share. On future connections to the same server, a client is able to use this share to encrypt early data.

Additionally, the pre-existing mechanism for Pre-Shared Keys (PSKs) has been extended to cover session resumption. This also requires a single round trip, and less computation than a full handshake. We describe its details when discussing PSKs and session resumption.

Record protection mechanisms. The earlier versions of TLS used the MAC-then-Encrypt general composition scheme as a record protection mechanism. Despite not being secure in general [9], the particular use of this scheme in SSL was shown to be safe in practice by Krawczyk [31]. While it is still used today in TLS 1.2, there was a proposal to replace it by the Encrypt-Then-MAC paradigm (cf. RFC 7366 [24]). Similarly, when Krawczyk [**?**] announced the OPTLS protocol on the TLS mailing list, he stated it would use Encrypt-then-MAC for record protection. Ultimately, the TLS working group decided that TLS 1.3 would avoid general composition schemes by only using block ciphers that can operate in so-called AEAD modes (Authenticated Encryption with Additional Data, cf. [40]). All non-AEAD ciphers have thus been removed in TLS 1.3.

Key derivation. In contrast to TLS 1.2, TLS 1.3 employs the use of handshake traffic keys as well as application traffic keys. This keying material is derived from two secrets, namely the ephemeral secret (`es`) and the static secret (`ss`). In the 1-RTT (EC)DHE handshake, the `es` and the `ss` are identical with the secret being derived from the ephemeral client and server key shares. In a PSK handshake, these two values are again identical and take on the value of the PSK. In PSK-DHE mode, the `es` is derived from the ephemeral client and server key shares and the `ss` is the PSK. In a 0-RTT handshake, the `es` is again derived from the ephemeral

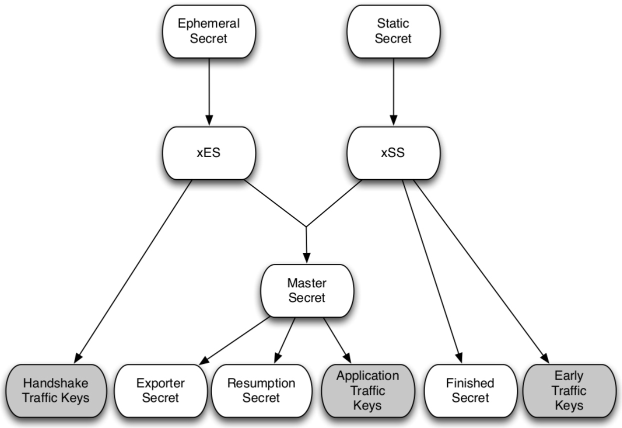

Figure 1. Key computation hierarchy for `draft-10`. Image from [48].

client and server key shares and the `ss` is computed using the server's semi-static key share and the client's ephemeral key share.

The secrets described above are also used as inputs to the HMAC-based construction, HKDF [21], [32] in order to derive a master secret `ms`, a resumption secret `rs` and a finished secret `fs`. These secrets are derived according to the schematic presented in Figure 1.

Another input to the HKDF is the `handshake_hash`. This consists of a hash of all the handshake messages, including all client and server messages, up to the present time but excluding the `Finished` messages. The final value of the `handshake_hash` is called the `session_hash`. As such, the session keys established are cryptographically bound to both of the shared secrets negotiated, and rely on both parties having a matching view of the handshake transcript.

2.2. New handshake modes

Some of the most significant changes in TLS 1.3 are due to the newly introduced handshake mechanisms. Here we provide a brief overview of these different modes, starting with a description of the regular, initial handshake.

Initial (EC)DHE handshake. The solid message flows in Figure 2 represent this handshake. Every protocol message followed by an asterisk can be omitted if only unilateral (server) authentication is required. Braces of the type { } indicate encryption under the handshake traffic key, whereas braces of the type [ ] indicate encryption under the application traffic key:

A client sends a server an offer of cryptographic parameters, including a client nonce, that are later used to establish session keys (`ClientHello`), and freshly generated Diffie-Hellman (DH) key shares along with the associated set of groups (`ClientKeyShare`). The server responds with its choice of cryptographic parameters, including a server nonce and a selected group from among those offered by the client (`ServerHello`). The server also sends its own freshly

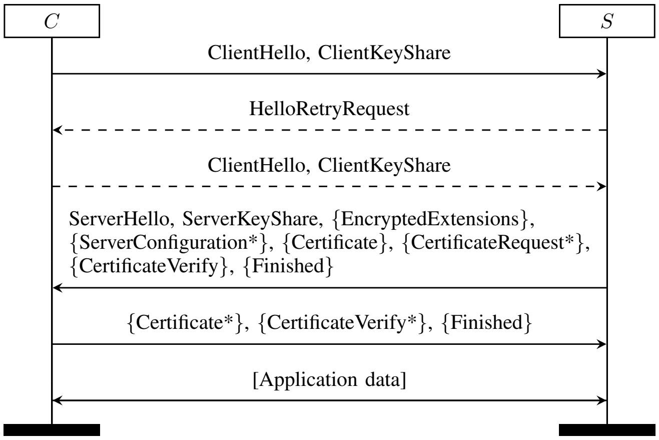
C
S
ClientHello, ClientKeyShare
HelloRetryRequest
ClientHello, ClientKeyShare
ServerHello, ServerKeyShare, {EncryptedExtensions},
{ServerConfiguration*}, {Certificate}, {CertificateRequest*},
{CertificateVerify}, {Finished}
{Certificate*}, {CertificateVerify*}, {Finished}
[Application data]

Figure 2. Full (EC)DHE handshake

generated DH key share (`ServerKeyShare`), extensions not used for key establishment (`EncryptedExtensions`) and a semi-static (EC)DH key share to be used in later handshakes (`ServerConfiguration`). Also included in the server's first flight are its public key certificate for authentication purposes (`Certificate`), an optional request for the client's certificate in the case that mutual authentication is desired (`CertificateRequest`), and a signature on all messages exchanged thus far (`CertificateVerify`). The server's `Finished` message comprises a MAC over the entire handshake using a handshake key derived from the DH key shares. Finally, if the client received a request for authentication, the client either sends its own certificate (`Certificate`) and a signature on the whole handshake thus far (`CertificateVerify`), or a blank certificate representing no authentication. As in the server's case, the client's `Finished` is a MAC over the entire handshake using a handshake key derived from the DH key shares. The purpose of the `Finished` messages is to provide integrity of the handshake as well as key confirmation.

If the client does not provide an appropriate key share in its first flight (it may provide groups that are unacceptable to the server, for instance), the server transmits a `HelloRetryRequest` message in order to entice the client to change its key share offer. Upon receipt of this message, the client should send a newly generated key share. These messages are indicated as dashed arrows in Figure 2. If no common parameters can be agreed upon, the server will send a `handshake_failure` or `insufficient security` alert and the session will be aborted.

0-RTT. Following the initial handshake in which the server provides the client with a semi-static (EC)DH share, the client is able to use this share to encrypt early data. Figure 3 depicts the 0-RTT handshake. The client's `EarlyDataIndication` value signals a 0-RTT handshake, which the server can choose to ignore (the server will not process the early data and a 1-RTT handshake will ensue). Braces of the type ( ) indicate encryption under the early traffic key derived from the server's semi-static key share and the client's ephemeral key share.

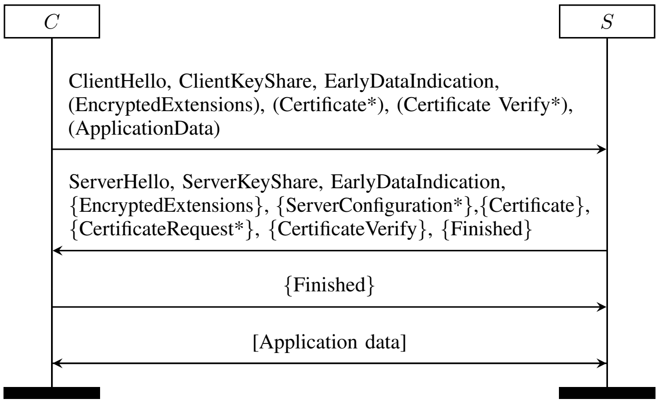
C
S
ClientHello, ClientKeyShare, EarlyDataIndication,
(EncryptedExtensions), (Certificate*), (Certificate Verify*),
(ApplicationData)
ServerHello, ServerKeyShare, EarlyDataIndication,
{EncryptedExtensions}, {ServerConfiguration*},{Certificate},
{CertificateRequest*}, {CertificateVerify}, {Finished}
{Finished}
[Application data]

Figure 3. 0-RTT handshake

PSKs and session resumption. TLS 1.3 effectively merges the PSK and session resumption functionalities of TLS 1.2 into a single handshake mode. There are two possible sources of PSKs: session tickets and out-of-band mechanisms. While the former is specified in `draft-10`, the latter has not yet been entirely clarified with regards to its intended implementation or assumed security properties. Figure 4 depicts a PSK handshake following an initial handshake. Note that a new session ticket gets sent by the server directly after receiving the client's `Finished` message in the initial handshake.

The client sends a key share in its first flight of the resumption handshake to allow for the server to decline resumption and fall back to the full (EC)DHE handshake. The `PreSharedKeyExtension` value indicates the identity of the PSK to be used in the exchange. We note that a PSK handshake need not only take the form of a resumption handshake. If a client and a server share an existing secret, a PSK handshake may be an initial handshake. PSKs may also be used in conjunction with an EC(DHE) exchange so as to provide forward secrecy; the corresponding mode is called PSK-DHE.

2.3. Stated goals and security properties

The TLS record protocol is claimed to provide confidentiality and integrity of application data. The TLS handshake protocol is claimed to allow unilateral or, optionally, mutual entity authentication, as well as establishing a shared secret that is unavailable to eavesdroppers and adversaries who can place themselves in the middle of the connection. The handshake is claimed to be reliable: no adversary can modify the handshake messages without being detected by the communicating parties.

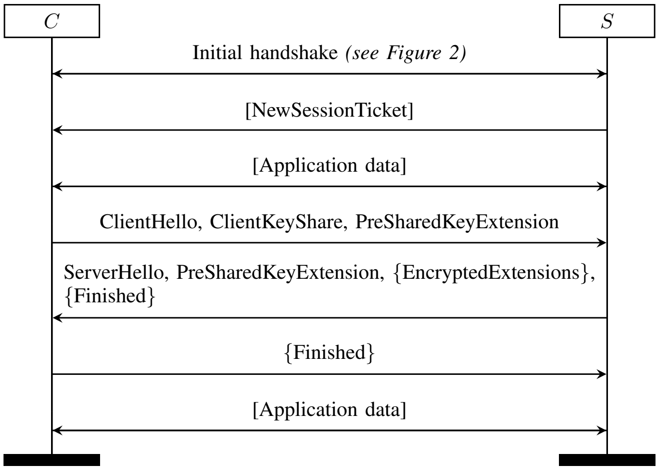
C
S
Initial handshake (see Figure 2)
[NewSessionTicket]
[Application data]
ClientHello, ClientKeyShare, PreSharedKeyExtension
ServerHello, PreSharedKeyExtension, {EncryptedExtensions},
{Finished}
{Finished}
[Application data]

Figure 4. PSK-resumption handshake (after an initial handshake)

The security properties thus inferred from `draft-10` include:

- unilateral authentication of the server (mandatory),
- mutual authentication (optional),
- confidentiality and perfect forward secrecy of session keys, and
- integrity of handshake messages.

These properties form the focus of our analysis. Table 1 outlines the full set of desired properties from "Appendix D: Security Analysis" of [47]. However, we note that this appendix contains the disclaimer "Todo: Entire security analysis needs a rewrite", and has not been updated since the major changes in `draft-7`, and hence we expect this set of properties to be updated in future revisions.

Security property	**Covered by analysis**	**Source**
Unilateral authentication (server)	Y	D.1.1
Mutual authentication	Y	D.1.1
Total anonymity	N	D.1.1
Confidentiality of ephemeral secret	Y	D.1.1*
Confidentiality of static secret	Y	D.1.1*
Perfect forward secrecy	Y	D.1.1.1
Integrity of handshake messages	Y	D.1.3
Protection of application data	N	D.2
Denial of service	N	D.3
Version rollback	N	D.1.2

* specification refers to outdated `master_secret`

TABLE 1. TLS 1.3 `DRAFT-10` PROPERTIES.

In addition to the stated goals, there are a few caveats which are encountered in the specification. For example, the 0-RTT application data and possible client authentication both come with a warning notice. In the 0-RTT handshake, the client is the only party to have provided freshness, therefore these early data messages may be replayed. In addition, the security of the early data depends on the semi-static (EC)DH share, which may have a considerable validity period, and therefore a large attack window. Therefore, early data cannot be considered to be forward secure.

3. Formally modelling the protocol behaviour

We build a formal model of the handshake and record protocols of `draft-10` in the framework of the Tamarin prover [50]. Tamarin is well-suited for this type of analysis for several reasons. First, Tamarin's multiset-rewriting semantics enable a direct specification of the complex state machines of TLS, including the complex interactions between all the handshakes, in a straightforward fashion. Second, its state-of-the-art support for Diffie-Hellman key exchange allows for a high degree of precision. Third, its property specification language (a fragment of first-order logic with quantification over time-points) lets us model the security properties intuitively and accurately.

3.1. Tamarin fundamentals

Here we provide a brief introduction to Tamarin. For a more detailed introduction, we suggest reading the Tamarin manual found at [51], the PhD thesis of Schmidt [49], or the PhD thesis of Meier [41].

Rules. The Tamarin semantics are based on multiset-rewriting. A Tamarin model defines a transition system whose state is a multiset of *facts*. The allowed transitions are specified by *rules*. At a very high level, Tamarin rules encode the behaviour of participants, as well as adversarial capabilities. In modelling cryptographic protocols, these rules play a role similar to oracles in Bellare-Rogaway style models.

Tamarin rules have a *left-hand side* (premises), *actions*, and a *right-hand side* (conclusions). The left-hand and right-hand sides of rules respectively contain multisets of facts. Facts can be *consumed* (when occurring in premises) and *produced* (when occurring in conclusions). Each fact can be either *linear* or *persistent* (marked with an exclamation point). While linear facts model limited resources that cannot be consumed more times than they are produced, persistent facts model unlimited resources, which can be consumed any number of times once they have been produced.

A rule can only be executed if its left-hand side can be matched with facts that are available for consumption in the current state. For instance, the `Fresh` rule depicted here

```
rule Fresh:
[ ]--[ ]->[ Fr(x) ]
```

has no premises or actions, and every execution of it produces a single linear `Fr(x)` fact. Note that only the `Fresh` rule can produce `Fr` facts, each of them unique.

Actions do not influence the transitions, but are "logged" when rules are triggered as a means of incrementally constructing observable action traces that in turn represent a record of a specific execution. As we will later see, actions (as part of action traces) form the glue between the defined transition system and the property specification language.

Cryptographic primitives. The ability to model cryptographic protocols requires the representation of cryptographic primitives. In Tamarin, symmetric encryption, for instance, is modelled using two binary functions, `senc` and `sdec`, and an equation of the form

$$\texttt{sdec(senc(m, k), k) = m},$$

where `k` is a shared secret key and `m` is a message. As certain primitives are used repeatedly across many cryptographic protocols, there are builtin definitions for them. The Tamarin *builtins* include equational theories for Diffie-Hellman group operations, asymmetric encryption, symmetric encryption, digital signatures and hashing.

The symmetric encryption builtin, for instance, could be used in this simple rule that models sending encrypted data out to a network:

```
rule Send:
[Fr(~k),Fr(~data)]--[Send(~data)]->[ Out(senc(~data,~k)) ]
```

The use of the builtin `Out` fact, as depicted in the `Send` rule, denotes that a message has been sent out to the network, i.e. `senc(~data,~k)` becomes known to the adversary. Receiving a message from the network is denoted by the corresponding `In` fact. In other words, `In(senc(~data,~k))` could form a premise of the rule which models receiving encrypted data from the network. Use of the $\sim$ symbol denotes a variable of the type `Fresh`. Other variable types include `Public`, denoted by $, and `Temporal`, denoted by #.

Security properties as lemmas. Tamarin's formulas are specified in a fragment of first-order logic and therefore offer the usual connectives (where `&` and `|` denote *and* and *or*, respectively), quantifiers `All` and `Ex`, and time-point ordering <. In formulas, the prefix `#` denotes that the following variable is of type *timepoint*. The expression `Action(args)@t` denotes that `Action(args)` is logged in the action trace at point `t`, resulting from an instantiation of a rule.

We use the Tamarin property language to encode different kinds of properties as *lemmas*. These can include, for example, basic state reachability tests as well as security properties. We follow the common Tamarin modelling approach, in which lemmas closely resemble properties as defined in Bellare-Rogaway models. For instance, where a Bellare-Rogaway definition will typically restrict the set of oracle queries the adversary can make (e.g., it cannot query for a decryption of the challenge ciphertext), in Tamarin we restrict the adversary in the statement of the lemma.

Axioms. The Tamarin tool also allows for the specification of *axioms*. These restrict the number of considered traces during analysis. For example, the following axiom instructs the Tamarin tool to only consider traces where all equality checks succeed:

```
axiom Equality_Checks_Succeed:
   "All x y #i. Eq(x,y) @ i ==> x = y".
```

We typically use axioms to avoid traces where:

- protocol participants initiate sessions with themselves,
- large numbers of key pairs are generated for a single protocol participant, or,
- unnecessary features appear when constructing traces without these features.

3.2. Constructing a protocol abstraction

The first step in the modelling process is to construct an abstraction of the handshake and record protocols which will in turn become the subject of our analysis. We attempt to strike a balance between a completely accurate, yet potentially complex model, and capturing only the most important cryptographic and algorithmic aspects of the protocols.

Perfect cryptography. We discussed the modelling of cryptography through builtins in Section 3.1. This results in the abstraction that our cryptographic primitives are *perfect*. For example, the encryption mechanism reveals nothing about the underlying plaintext.[2] Similarly, we assume:

- signatures are unforgeable,
- hash functions act as random oracles (with zero collision probability),
- MACs are unforgeable, and,
- all parties generate truly random values.

This is one of the possibilities for extending our analysis; while the builtins assume perfect cryptography, we can easily weaken these primitives by introducing rules which, for example, let the adversary create signature and MAC forgeries.

Configuration parameters. We also note that we simplify our model by treating all parameters that do not directly influence the security of the protocol as abstract quantities within the model. For example, the `EncryptedExtensions` message of a client in the 0-RTT handshake will be logically bundled together with all other messages of this kind and represented by the single public value `exts`.

However, since these components comprise part of the handshake transcript, we establish whether the client and server agree on these values by the end of the handshake through the transcript integrity property.

Alert messages. We also do not explicitly model the TLS alert protocol; our model does not capture errors arising from deviations in the protocol that would result in the immediate termination of a connection (fatal alerts), or acknowledgements of a graceful shutdown (closure alerts).

From the perspective of our model, and the security properties we are capturing, an alert and subsequent connection closure is equivalent to a trace which simply does not have any subsequent rules for that state.

Over-approximations. In certain situations, we assume that the client/server will send the maximal message load. For example, the client will always send 0-RTT data. Similarly, we model the server as always including a `CertificateRequest` message in the first flight. However, the client does not always send authentication parameters, and the server does not necessarily accept these parameters if sent. Therefore, the possible traces we observe are equivalent to those in which the server optionally sends the request.

3.3. Encoding our abstract model in the Tamarin framework

The second step in our modelling involves encoding the constructed abstract model as Tamarin rules. At a very high level, rules capture honest party and adversary actions alike. In the case of legitimate clients and servers, our constructed model rules generally correspond to all processing actions associated with respective flights of messages.

For instance, our first client rule captures a client generating and sending all necessary parameters as part of the first flight of an (EC)DHE handshake, as well as transitioning to the next client state within the model. In Figure 5, the `let...in` block allows us to perform basic variable substitutions. In practice, this is useful for enforcing the type of variables, such as `C = $C`, or for keeping the rule computations logically separated. We use the variable `tid` to name the newly created client thread. The action `DH(C,~a)` allows us to map the private DH exponent `~a` to the client `C`. The `Start(tid, C, 'client')` action signifies the instantiation of the client `C` in the role of `'client'` and the `Running(C, S, 'client', nc)` action indicates that the client `C` has initiated a run of the protocol with the server `S`, using the fresh value `nc` as what is known as the `client_random` value in TLS. The `C1` action simply marks the occurrence of the `C_1` rule with its associated `tid`. The `St_C_1_init` fact encodes the local state of thread `tid`, which doubles as a program counter by allowing the client to recall sending the first message in thread `tid`. The `Out` fact represents sending the first client message to the network, whereafter it becomes adversarial knowledge.

Overview of client handshake rules. Figures 6 and 7 capture all relevant model rules and represent the union of all the options that a client and a server have in a single execution. We explain the client-side behaviour and map it to the corresponding transitions while briefly mentioning the intended server interaction. The client can initiate three types of handshake: an (EC)DHE handshake (`C_1`), a PSK handshake (`C_1_PSK`) and a 0-RTT handshake (`C_1_KC`); we use KC (an abbreviation for Known Configuration) to denote 0-RTT handshakes. In the (EC)DHE handshake, the server may reject the client parameters due to a possible mismatch, whereafter the client needs to provide new parameters (`C_1_retry`). Additionally, the client may optionally authenticate in the 0-RTT case (`C_1_KC_Auth`). While the (EC)DHE and 0-RTT handshakes only have a single

[2] Our current analysis therefore does not cover Logjam-style attacks [2].

```
rule C_1:
let
    // Default C1 values
    tid = ~nc

    // Client Hello
    C   = $C
    nc  = ~nc
    pc  = $pc
    S   = $S

    // Client Key Share
    ga   = ’g’^~a

    messages = <nc, pc,ga>
in
    [ Fr(nc)
    , Fr(~a)
    ]
  --[ C1(tid)
    , Start(tid, C, ’client’)
    , Running(C, S, ’client’, nc)
    , DH(C, ~a)
    ]->
    [ St_C_1_init(tid, C, nc, pc, S,
                   ~a, messages, ’no_auth’)
    , Out(<C,nc, pc,ga>)
    ]
```

Figure 5. Rule C_1 in our Tamarin model of TLS 1.3 `draft-10`

continuation (respectively `C_2` and `C_2_KC`), the PSK handshake has two different modes: plain PSK (`C_2_PSK`) and PSK with DHE (`C_2_PSK_DHE`). The latter is used to obtain PFS guarantees by means of adding an ephemeral (EC)DH value to the applicable key derivations.

We model the server as always requesting client authentication, but allow traces to capture when the server accepts authentication or not. If the client decides to authenticate, it sends the authentication messages along with the client `Finished` message (`C_2_Auth`). Otherwise, only the `Finished` message is transmitted (`C_2_NoAuth`). The handshake concludes with the client either receiving a new session ticket (`C_3_NST`) or doing nothing (`C_3`). The session ticket can be used for resumption in a later PSK handshake. The client can then proceed to send (`C_Send`) and receive (`C_Recv`) any finite number of application data messages.

We note that the figure represents a ‘snapshot’ in time beyond the initial establishment of a connection. Consequently, it lacks certain states and transitions in which the client initiates or the server responds to a newly established connection. We list the omissions here as rules that can follow a previously executed rule:

- C_2→C_1_KC
- C_2_KC→C_1_KC
- S_1→S_1_KC
- S_1_KC→S_1_KC
- S_1→S_1_KC_RecvAuth
- S_1_KC→S_1_KC_Recvauth
- C_3_NST→C_1_PSK
- S_3_NST→S_1_PSK
- S_3_NST→S_1_PSK_DHE

Managing model complexity. The complexity of TLS 1.3 presents an interesting challenge for automated symbolic analysis. As Figures 6 and 7 demonstrate, the introduction of new handshake modes has dramatically increased the number of state transitions in comparison to TLS 1.2.

In software engineering, conditional branches are a fairly mundane part of code. For example, the code might perform the check: “`if` received client authentication `then` verify signature and set client status to authenticated `else` do nothing”. However, in Tamarin we require two distinct state transitions representing these two possibilities.

The TLS handshake exhibits such conditional branching. Ideally, branches would be represented by as few rules as possible, which can be done by merging some of the resulting states into one. For example, by the end of the server’s first phase, the state needs to contain a transcript of the received messages, the computed values of `ss` and `es`, and the authentication status of the client.

While all four handshake modes will compute these values in a different way, from the point of computation onwards the server’s behaviour does not depend on the handshake mode. Therefore these can be merged into the resulting s_1 state.

With this approach, we can create simple rules that ensure the composability of the various protocol modes and closely follow the original specification. For example, the numbering of states (c_1, c_2, etc.) corresponds to message flights. In some cases, we require two rules to construct a single message flight, e.g. `C_2` and `C_2_Auth`, wherein a client optionally adds a signature to the final client handshake message.

3.4. Examples of complex interactions

By defining the client and server rules as outlined above, we now have the ability to model the interaction of an unbounded number of interleaved handshakes. That is, while we express properties in terms of a client and a server, there may exist an unbounded number of other interacting agents, which the adversary may additionally compromise through revealing their long-term keys. The adversary can then impersonate these agents, leading to an increase in the number of possible interactions.

Consider the following scenario : A client and a server have derived session keys after agreeing to use a PSK. We know that at some point the client must have authenticated the server (assuming the PSK is not from the out-of-band mechanism). However, we potentially need to resolve an unbounded number of handshakes before we arrive at the initial handshake in which the client verified the server’s signature. The Tamarin prover allows us to reason inductively about such scenarios, facilitating the verification of important security properties that are typically out of reach of backwards unfolding.

Our Tamarin model is available for inspection at [1].

4. Formal analysis of the model

In this section, we provide the details of our analysis. In particular, we describe our threat model, the required security properties and how we formally model them in Tamarin. We

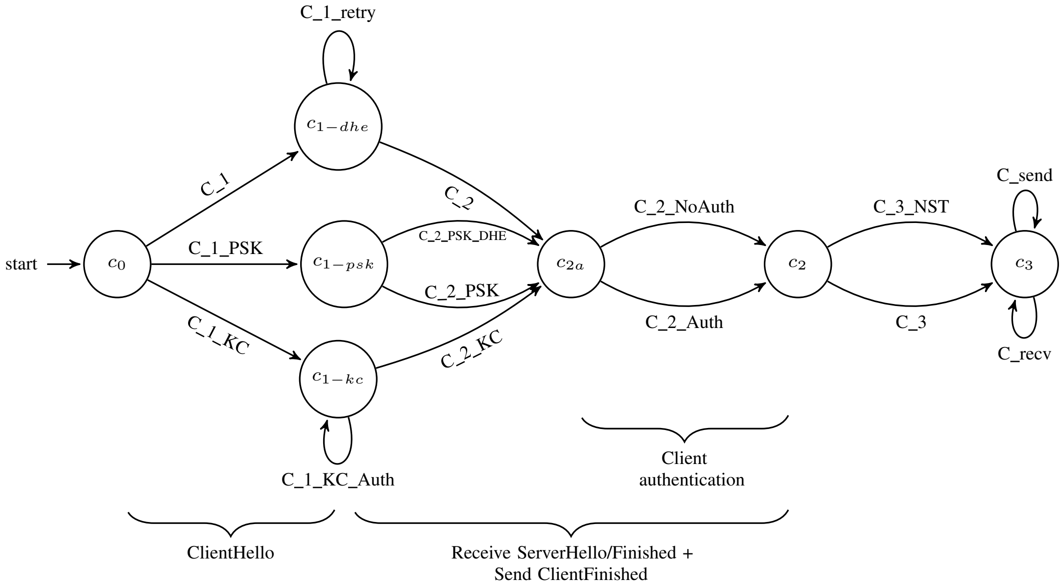

Figure 6. Partial client state machines for `draft-10` as modelled in our Tamarin analysis. The diagram represents the union of all the options for a client in a single execution. Not depicted are the additional transitions representing a client starting a new handshake using either a PSK established by `C_3_NST` or a `ServerConfiguration` for a 0-RTT handshake. Note that the `C_1_KC_Auth` edge may only occur once per handshake.

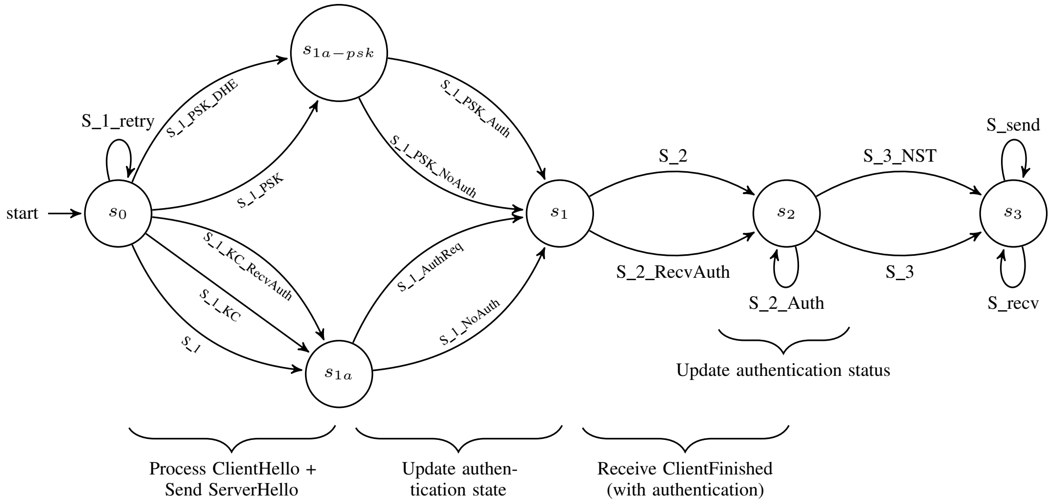

Figure 7. Partial server state machines for `draft-10` as modelled in our Tamarin analysis. The diagram represents the union of all the options for a server in a single execution. Not depicted are the additional transitions representing a server starting a new handshake using either a PSK established by `S_3_NST` or a `ServerConfiguration` for a 0-RTT handshake. Note that the `S_2_Auth` edge may only occur at most once per handshake.

then give our analysis results, reflect on our findings and provide recommendations for the TLS Working Group.

4.1. General approach and threat model

Our aim is to analyse the core security properties of the TLS 1.3 protocol. The work on TLS 1.3 to date generally considers subprotocols in isolation. Our work, as explained in the previous section, also considers all the possible complex interactions between the various subprotocols. For this interaction, we prove both secrecy and authentication properties.

The threat model that we consider in our analysis is an active network adversary that can compromise the long-term keys of agents. In particular, we consider adversaries that can compromise the long-term keys of all agents after the thread under attack ends (to capture PFS) as well as the long-term keys of agents that are not the actor or the intended peer of the attacked thread, at any time (to capture Lowe-style MITM attacks and to contain the consequences of long-term key compromise). Moreover, we include limited support for Actor Key Compromise [8] by allowing the adversary to reveal the long-term keys of the client when verifying the unilateral authentication properties and the secrecy of early data keys.

Similarly to standard AKE models, our threat model has two components: the Tamarin rule that encodes the full capability (i.e., the ability to compromise an agent's long-term private key) and a restriction on the security notion that prevents the adversary from compromising all the keys (corresponding to the fresh/clean predicates in AKE models). We give the rule here, and return to the restrictions when we describe the specific properties.

```
rule Reveal_Ltk:
  [ !Ltk($A, ~ltkA) ] --[ RevLtk($A) ]-> [ Out(~ltkA) ]
```

This rule can be triggered if a long-term private key `~ltkA` was previously generated for the agent `$A`. The right-hand side of the rule encodes that `~ltkA` is sent on the network, effectively adding it to the adversary's knowledge. Additionally, we log the action `RevLtk($A)`, which will enable us to restrict this capability in the property specifications.

The model described in this and the previous section describes the behaviour of the TLS 1.3 protocol in the presence of an active network adversary. The Tamarin model assumes the standard black-box cryptography assumption, as outlined in Section 3.2. This view simplifies the proofs and enables the analysis of many different security contexts.

In the two following sections, we model and verify the required secrecy and authentication properties.

4.2. Secrecy properties and results

We formally model and analyse two main secrecy properties. The first is the secrecy of session keys that implies perfect forward secrecy in the presence of an active adversary. The formal property that we verify is:

```
lemma secret_session_keys:
(1) "All actor peer role k #i.
(2)  SessionKey(actor, peer, role, <k, 'authenticated'>)@i
(3)  & not ((Ex #r. RevLtk(peer)@r  & #r < #i)
           |(Ex #r. RevLtk(actor)@r & #r < #i))
(4)  ==> not Ex #j. KU(k)@j"
```

Intuitively, the above property requires that for all protocol behaviours and for all possible values of the variables on the first line (`All`) (1): if an authenticated session key `k` is accepted (encoded by the occurrence of the `SessionKey` action) (2), and the adversary has not revealed the long-term private keys of the actor or the peer before the session key is accepted (3), then the adversary can not derive the key `k` (4).

Our way of modelling this property is very flexible. In the unilaterally authenticated mode, only the client establishes a session key with the flag `authenticated`. In the mutually authenticated mode, both roles log this action. Both properties can be used to verify the appropriate secrecy properties. As we will see later, this is also suitable for the more flexible delayed client authentication modes that will be allowed in the final TLS 1.3 specification.

The second property that we prove is that the clients' early data keys are secure as long as the long-term private key of the server is not revealed.

```
lemma secret_early_data_keys:
(1) "All actor peer k #i.
(2)  EarlyDataKey(actor, peer, 'client', k)@i
(3)  & not ((Ex #r. RevLtk(peer)@r))
(4)  ==> not Ex #j. KU(k)@j"
```

In particular, each time (1) that a client logs that it has produced an early data key (2) and the peer's long-term private keys are not compromised (3), then the adversary does not know the early data key (4).

Proof approach in Tamarin. Many of the security properties of TLS stem from the secrecy of the shared secrets, i.e. the ephemeral secret (`es`) and the static secret (`ss`). Proving the secrecy of these components initially seems simple; at its core, the main TLS mechanism includes an authenticated Diffie–Hellman exchange. However, complications arise due to the interactions between different handshake modes in an unbounded number of sessions and connections, and powerful adversarial interference.

As a first step, it is necessary to prove a few fundamental invariant properties. These help all future proofs by either reducing the number of contradictory dead-ends which the prover would otherwise explore; or to help skip some common intermediate steps. In particular, it is essential to apply some straightforward inductive proofs to avoid falling into the many infinite loops present.

From here, atomic auxiliary lemmas can be constructed. These lemmas help us piece together the more complicated proofs in a modular way. For example, a common deduction uses the fact that knowledge of the PSK implies that the adversary must also have knowledge of some (`ss`, `es`) pair from a previous handshake.

Ideally, the auxiliary lemmas are sufficiently small and incremental that they can be proved automatically. Since each describes a small property which is likely to remain consistent throughout model changes, these can be used to quickly incorporate changes and reproduce proofs. The proofs for secrecy of `ss` and `es` follow from the auxiliary lemmas in a more manageable way than would otherwise be the case. The main burden of proof is to unravel the client and server states to a point where the adversary needs to break the standard Diffie-Hellman assumptions, or else the secrecy follows from inductive reasoning.

Finally, the proof of session-key secrecy then follows from the secrecy proofs for the `ss` and `es` values, which are both used as key-derivation inputs. Using this approach, we successfully verify these properties in Tamarin for the full interaction between the modes modelled.

We note that the construction of the auxiliary lemmas and the proving of the secrecy of `ss` and `es` requires an intimate knowledge of TLS 1.3 an great deal of ingenuity; this part of the analysis is not a straight-forward application of the Tamarin tool. A great deal of interaction with the tool is required so as to correctly guide it through the proof trees of the respective `ss` and `es` lemmas.

Separation of properties. One of the decisions made when specifying the security properties was to separate the secrecy and authentication requirements. Note that we could have equally combined both into a single property, as commonly defined in AKE models.

The benefit of this approach is twofold. First of all, separating the properties results in a richer understanding of the security of the protocol. For example, the structure of the proof confirms the intuition that the secrecy of session keys depends largely on the use of a Diffie-Hellman exchange.

The second benefit of this approach is to provide a better foundation for future analysis. While our current model considers the security of all handshake modes equally, there are some discrepancies in the guarantees provided by the various handshake modes. For example, if we were to allow adversarial compromise of semi-static secrets such as PSKs and server semi-static DH exponents, we would discover that the secrecy of the static secret is not immediate in all handshake modes. By keeping the properties separate, it will be easier to move to a more nuanced security model in the future.

4.3. Authentication properties and results

We model authentication properties as agreement on certain values, such as agent identities and nonces. This is one of the main ways of defining authentication [37].

The first property that we model is that when a client assumes there is a peer with whom it shares nonces, then this is actually the case. The concrete formula is the following:

```
lemma entity_authentication:
(1) "All actor peer nonces #i.
(2)  CommitNonces(actor, peer, ’client’, nonces)@i
(3)  & not (Ex #r. RevLtk(peer)@r)
(4)  ==> (Ex #j peer2.
(5)  RunningNonces(peer, peer2, ’server’, nonces)@j
(6)  & #j < #i)"
```

In detail, it specifies that when the client logs that it has observed certain nonces at the end of its thread and thinks it is communicating with a specific peer (1,2), and the long-term private key of this peer was not compromised (3), then there existed a thread (4) of that peer in the server role that agrees on the nonces (5) earlier (6).

Note that we would like to perhaps verify that `peer2` is equal to `actor`, but in the unilaterally authenticated mode, no such guarantee can be obtained.

The second property encodes that not only do the actor and the peer agree on the nonces and who the server is, they in fact agree on the complete transcript.

```
lemma transcript_agreement:
"All actor peer transcript #i.
 CommitTranscript(actor, peer, ’client’, transcript)@i
 & not (Ex #r. RevLtk(peer)@r)
 ==> (Ex #j peer2.
 RunningTranscript(peer, peer2, ’server’, transcript)@j
 & #j < #i)"
```

The above two properties only provide guarantees for the client, as in the main use case where only the server is authenticated.

We now turn to the (optional) server guarantees. We have equivalent properties in the mutually authenticated case. However, since both parties authenticate each other, we can achieve a stronger notion of authentication, but with the restriction that the adversary cannot reveal either long-term key.

The third property represents the authentication guarantee for the server, which can be obtained if the server performs the mutually authenticated handshake or requests client authentication.

```
lemma mutual_entity_authentication:
"All actor peer nonces #i.
 CommitNonces(actor, peer, ’server’, nonces)@i
 & not ((Ex #r. RevLtk(peer)@r)
       |(Ex #r. RevLtk(actor)@r))
 ==> (Ex #j.
 RunningNonces(peer, actor, ’client’, nonces)@j
 & #j < #i)"
```

The fourth property is analogous to the second, and ensures that the server obtains a guarantee on the agreement on the transcript with the client, after it has been authenticated.

```
lemma mutual_transcript_agreement:
"All actor peer transcript #i.
 CommitTranscript(actor, peer, ’server’, transcript)@i
 & not ((Ex #r. RevLtk(peer)@r)
       |(Ex #r. RevLtk(actor)@r))
 ==> (Ex #j.
 RunningTranscript(peer, actor, ’client’, transcript)@j
 & #j < #i)"
```

Implicit authentication. In building the series of lemmas which lead to the final security properties, the most problematic areas coincided with the PSK modes. In particular, the security of the PSK handshake relies on knowing that the resumption secret can only be known by a previous communication partner. This is the implicit authentication property.

While we were able to overcome this challenge and eventually prove that this property holds, it does identify a potentially troublesome component to analyse. As we will see in the next section, the PSK mode certainly requires close attention.

We note that there are a plethora of entity authentication algorithms which could be used to add an explicit authentication step to the session resumption section, some of which can be found in [25].

4.4. Analysis conclusions

Our model from Section 3 covers many possible complex interactions between the various modes, for an unbounded number of sessions. When combined with the security properties in this section, this gives rise to very complex verification problems. Nevertheless, we managed to successfully prove the main properties. Our results imply the absence of a large class of attacks, many of which are not covered by other analysis methods, especially attacks that exploit the interaction between the various modes. This is a very encouraging result, since it shows that the core design underlying `draft-10` is solid.

Despite this, the late addition of new functionalities can still be problematic, as will become clear in the next section.

5. Enabling client authentication in PSK mode

While `draft-10` does not yet appear to permit certificate-based client authentication in PSK mode (and in particular in resumption using a PSK), we extended our model as specified in one of the proposals for this intended functionality [45].

By enabling client authentication either in the initial handshake, or with a post-handshake signature over the handshake hash, our Tamarin analysis finds an attack. The result is a violation of client authentication, as the adversary can impersonate a client when communicating with a server.

5.1. The attack

We note that the attack as described here is for the delayed authentication setting, but can easily be adapted for authentication as part of the handshake.

We now describe the attack depicted in Figure 8 in more detail: Alice plays the role of the victim client, and Bob the role of the targeted server. Charlie is an active man-in-the-middle adversary, whom Alice believes to be a legitimate server. In the interest of clarity we have omitted message components and computations which are not relevant to the attack. The full attack can be reproduced using our code at [1].

The attack proceeds in three main steps, each involving different TLS subprotocols.

Step 1: Establish legitimate PSKs. In the first stage of the attack, Alice starts a connection with Charlie, and Charlie starts a connection with Bob. In both connections, a PSK is established. At this point, both handshakes are computed honestly. Alice shares a PSK denoted PSK_1 with Charlie, and Charlie shares a PSK denoted PSK_2 with Bob.

Note that Charlie ensures the session ticket (psk_id) is the same across both connections by replaying the value obtained from Bob.

Step 2: Resumption with matching freshness. In the next step, Alice wishes to resume a connection with Charlie using PSK_1. As usual, Alice generates a random nonce nc, and sends it together with the PSK identifier, psk_id.

Charlie re-uses the value nc to initiate a PSK-resumption handshake with Bob, using the same identifier, psk_id. Bob responds with a random nonce ns, and the server `Finished` message, computed using PSK_2.

Charlie now re-uses the nonce ns, and recomputes the server `Finished` message using PSK_1. Alice returns her `Finished` message to Charlie, who recomputes it using PSK_2.

At this point, Alice and Charlie share session keys (i.e., application traffic keys) derived from PSK_1, and Charlie and Bob share session keys derived from PSK_2. Note that the keys that Charlie shares with Alice and with Bob respectively, are distinct.

Step 3: Delayed client authentication. Following the resumption handshake, Charlie attempts to make a request to Bob over their established TLS channel. The request calls for client authentication, so Charlie is subsequently prompted for his certificate and verification[3]. Charlie re-encrypts this request for Alice.

To compute the verification signature, Alice uses the `session_hash` value, which is defined as the hash of all handshake messages excluding `Finished` messages. In particular, the session hash will contain nc, ns, and the session ticket psk_id.

Notice that this session hash will match the one of Charlie and Bob. Therefore, this signature will also be accepted by Bob. Hence, Charlie re-encrypts the signature for Bob, who accepts Alice's certificate and verification as valid authentication for Charlie.

Charlie has therefore successfully impersonated Alice to Bob, and even has full knowledge of the session keys. This enables Charlie to impersonate Alice in future communication with Bob, allowing him to fake messages or to access confidential resources, for instance, and violate the secrecy of messages that Bob tries to send to Alice. Thus, the attack completely breaks client authentication.

5.2. Underlying cause and mitigation

The above attack is possible due to the absence of a strong binding of the client signature to the server identity. Therefore, the attacker is able to reuse the signature it receives to impersonate the client to a server. The second component of the attack is that the attacker is able to force the two resumption sessions to have matching transcripts.

This suggests several potential ways to mitigate the attack. The most direct route would be to include the server certificate in the handshake hash. A similar fix is done in the 0-RTT case, where the server certificate is bound to the semi-static DH share. However, this is not ideal, because it complicates the out-of-band mechanism.

Another potential route might be to implement an explicit authentication step as part of the PSK mechanism, as suggested in Section 4.3.

In parallel to our analysis, the TLS Working Group has proposed several modifications to `draft-10` in the move towards `draft-11`. One of these proposals is PR#316 [46] (which takes a different approach to [45]), which explicitly allows client authentication in the context that we analyse. Additionally, PR#316 redefines the client signature based on a new Handshake Context value, which includes the server `Finished` message. Intuitively, this new definition appears to address the attack because the adversary will need to force the `Finished` messages to match across the two connections. However, the `Finished` message is bound to the PSK, which is derived from a previously authenticated session, whether using certificates or out-of band mechanisms.

Our discussions with members of the TLS Working Group reveal that they were previously not aware of the possibility of our attack, and the resulting strict necessity for a stronger binding between the client certificate and the security context that emerges from combining the PSK mode with client authentication.

[3] This is one of the main use cases for the delayed client authentication mode [45].

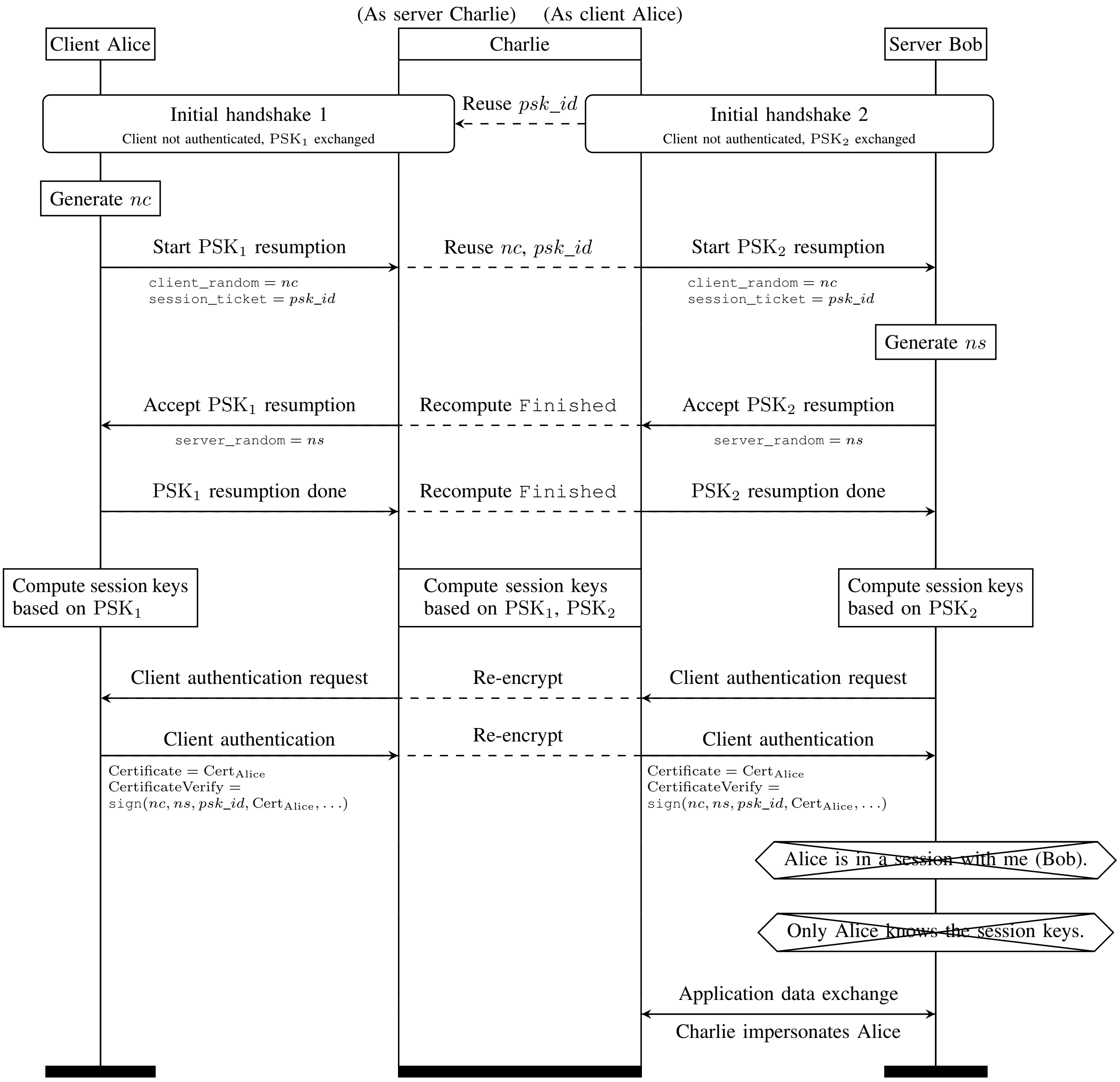
(As server Charlie)
(As client Alice)
Client Alice
Charlie
Server Bob
Initial handshake 1
Client not authenticated, PSK_1 exchanged
Reuse psk_id
Initial handshake 2
Client not authenticated, PSK_2 exchanged
Generate nc
Start PSK_1 resumption
client_random $= nc$
session_ticket $= psk\_id$
Reuse nc, psk_id
Start PSK_2 resumption
client_random $= nc$
session_ticket $= psk\_id$
Generate ns
Accept PSK_1 resumption
server_random $= ns$
Recompute Finished
Accept PSK_2 resumption
server_random $= ns$
PSK_1 resumption done
Recompute Finished
PSK_2 resumption done
Compute session keys based on PSK_1
Compute session keys based on PSK_1, PSK_2
Compute session keys based on PSK_2
Client authentication request
Re-encrypt
Client authentication request
Client authentication
Certificate $= \mathrm{Cert}_{\mathrm{Alice}}$
CertificateVerify $=$
sign$(nc, ns, psk_id, \mathrm{Cert}_{\mathrm{Alice}}, \ldots)$
Re-encrypt
Client authentication
Certificate $= \mathrm{Cert}_{\mathrm{Alice}}$
CertificateVerify $=$
sign$(nc, ns, psk_id, \mathrm{Cert}_{\mathrm{Alice}}, \ldots)$
Alice is in a session with me (Bob).
Only Alice knows the session keys.
Application data exchange
Charlie impersonates Alice

Figure 8. Client impersonation attack on TLS 1.3 `draft-10` if delayed client authentication allowed in PSK mode. The attack involves two handshakes, two resumptions, and a client authentication request. A full explanation is given in Section 5.1.

6. Conclusions and future work

We model TLS 1.3 draft-10 in the Tamarin prover framework. We verify the core security properties of the protocol in a complex setting where an unbounded number of concurrent connections interact in various protocol modes. We consider the unilateral and mutual authentication modes with respect to an active Dolev-Yao adversary capable of long-term key reveal, and consider perfect forward secrecy. We provide the first supporting evidence for the security of several complex protocol mode interactions. Our results imply the absence of a large class of attacks, many of which are not covered by other analysis methods, especially attacks that exploit the interaction between the various modes. This is a very encouraging result, since it shows that the core design underlying draft-10 is solid.

However, we find an attack on the delayed client authentication mechanism if it can be combined with a PSK-resumption handshake along the lines of the proposal in [45]. The attack that we find showcases the complex interactions that are covered by our Tamarin analysis: the attack involves two initial handshakes, two PSK-resumption handshakes, and a client authentication exchange in which the adversary re-encrypts messages.

Recommendations. Since the TLS Working Group aims to include the functionality exploited by the attack, steps must be taken to ensure the resulting mechanism is secure. Our analysis suggests that some proposed changes to the signature contents (cf. PR#316, [46]) can prevent the attack. The TLS Working Group has confirmed that they were unaware that including the Finished messages in the client authentication was a strict requirement to prevent this attack. The question remains if the proposed remedy of adding the Finished messages to the handshake context of the client certificate is indeed effective for prevent all such attacks. While we plan to give a conclusive answer to this question in the near future, we currently wish to support the suggested inclusion.

We believe that a complete state machine, which would contain considerably more states and transitions to account for all the possible side-cases contained in draft-10, should be added to the final specification to avoid any kind of ambiguity when the protocols will be implemented or analysed. Such ambiguities can lead to incorrect implementations and the inadvertent introduction of serious vulnerabilities, as is pointed out in [10].

Future work. Our Tamarin model is at the stage where it allows for checking both that basic security properties are maintained, and that further tweaks and new functionalities achieve their goals. This places us in a position of being able to provide value feedback to the TLS Working Group with regards to suggested protocol changes and consequently helps to accelerate the development process. Our rigorous analysis provides additional confidence in draft-10 and its successors, which is critical for such an important and elaborate protocol design.

Our analysis cannot cover *all* aspects of TLS 1.3. For example, the treatment of specific ciphers is beyond the scope of our current model. This is also why our work is just one part of a larger, concerted effort to use all kinds of different approaches in investigating the many different facets of TLS 1.3.

We would also like to note that the protocol has not yet been entirely finalised. However, many of the announced changes, such as reordering the arguments to hashes or merging/splitting some extensions, can be rapidly integrated into our analysis, mostly thanks to our flexible modelling and automation support for a large number of proofs. This enables us to continue to help shape and support the choices in the design of TLS 1.3 through to its completion.

References

[1] Archive with TLS 1.3 Rev 10 models and property specifications for the Tamarin prover, 2015. http://tls13tamarin.github.io/TLS13Tamarin/.

[2] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thom, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Bguelin, and P. Zimmermann. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice, 2015. https://weakdh.org/imperfect-forward-secrecy.pdf.

[3] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the Security of RC4 in TLS. In *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 305–320, 2013.

[4] N. J. AlFardan and K. G. Paterson. 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*, 2012.

[5] N. J. AlFardan and K. G. Paterson. 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*, pages 526–540, 2013.

[6] G. V. Bard. The Vulnerability of SSL to Chosen Plaintext Attack. *IACR Cryptology ePrint Archive*, 2004:111, 2004.

[7] G. V. Bard. 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*, pages 99–109, 2006.

[8] D. Basin, C. Cremers, and M. Horvat. Actor key compromise: Consequences and countermeasures. In *Proc. of the 27th IEEE Computer Security Foundations Symposium (CSF)*, 2014.

[9] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In T. Okamoto, editor, *Advances in Cryptology ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer Berlin Heidelberg, 2000.

[10] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue. 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*, pages 535–552, 2015.

[11] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P. Strub. 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*, pages 98–113, 2014.

[12] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu. Verified Cryptographic Implementations for TLS. *ACM Trans. Inf. Syst. Secur.*, 15(1):3, 2012.

[13] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P. Strub. Implementing TLS with Verified Cryptographic Security. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 445–459, 2013.

[14] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and S. Z. Béguelin. Proving the TLS Handshake Secure (as it is). *IACR Cryptology ePrint Archive*, 2014:182, 2014.

[15] D. Bleichenbacher. 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*, pages 1–12, 1998.

[16] B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux. 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*, pages 583–599, 2003.

[17] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Informational), August 2008.

[18] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. 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*, pages 1197–1210, 2015.

[19] T. Duong and J. Rizzo. Here Come the ⊕ Ninjas. Unpublished manuscript, May 2011.

[20] T. Duong and J. Rizzo. The CRIME Attack. Ekoparty Security Conference presentation, 2012.

[21] P. Eronen and H. Krawczyk. HMAC-based Extract-and-Expand Key Derivation Function (HKDF), May 2010. Available at https://tools.ietf.org/html/rfc5869.

[22] M. Fischlin and F. Günther. Multi-stage key exchange and the case of google's QUIC protocol. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1193–1204, 2014.

[23] F. Giesen, F. Kohlar, and D. Stebila. On the security of TLS renegotiation. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 387–398, 2013.

[24] P. Gutmann. Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7366 (Informational), September 2014.

[25] International Organization for Standardization, Genève, Switzerland. Information technology – Security techniques – Entity authentication – Part 2: Mechanisms using symmetric encipherment algorithms. ISO/IEC 9798-2:2008/Cor 3:2013, 2013.

[26] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. A Standard-Model Security Analysis of TLS-DHE. *IACR Cryptology ePrint Archive*, 2011:219, 2011.

[27] T. Jager, J. Schwenk, and J. Somorovsky. 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*, pages 1185–1196, 2015.

[28] J. Jonsson and B. S. Kaliski Jr. On the Security of RSA Encryption in TLS. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 127–142, 2002.

[29] V. Klíma, O. Pokorný, and T. Rosa. 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*, pages 426–440, 2003.

[30] M. Kohlweiss, U. Maurer, C. Onete, B. Tackmann, and D. Venturi. (De-)Constructing TLS. *IACR Cryptology ePrint Archive*, 2014:20, 2014.

[31] H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 310–331, 2001.

[32] H. Krawczyk. 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*, pages 631–648, 2010.

[33] H. Krawczyk, K. G. Paterson, and H. Wee. On the Security of the TLS Protocol: A Systematic Analysis. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 429–448, 2013.

[34] H. Krawczyk and H. Wee. The OPTLS Protocol and TLS 1.3. *IACR Cryptology ePrint Archive*, 2015:978, 2015.

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

[36] Y. Li, S. Schäge, Z. Yang, F. Kohlar, and J. Schwenk. 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*, pages 669–684, 2014.

[37] G. Lowe. A Hierarchy of Authentication Specifications. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, CSFW '97, pages 31–, Washington, DC, USA, 1997. IEEE Computer Society.

[38] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. How Secure and Quick is QUIC? Provable Security and Performance Analyses. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 214–231, 2015.

[39] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. 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*, pages 62–72, 2012.

[40] D. McGrew. An Interface and Algorithms for Authenticated Encryption. RFC 5116 (Informational), January 2008.

[41] S. Meier. Advancing automated security protocol verification, 2013. Doctoral thesis, ETH Zurich, Switzerland.

[42] P. Morrissey, N. P. Smart, and B. Warinschi. A Modular Security Analysis of the TLS Handshake Protocol. *IACR Cryptology ePrint Archive*, 2008:236, 2008.

[43] K. G. Paterson, T. Ristenpart, and T. Shrimpton. Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 372–389, 2011.

[44] L. C. Paulson. Inductive Analysis of the Internet Protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.

[45] A. Popov. TLS 1.3 client authentication. In *Meeting proceedings of the IETF-93 Workshop, Prague*. Retrieved from https://www.ietf.org/proceedings/93/slides/slides-93-tls-2.pdf, 2015.

[46] E. Rescorla. TLS 1.3 specification pull request: Wip client auth revision #316. https://github.com/tlswg/tls13-spec/pull/316/.

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

[48] E. Rescorla. TLS 1.3 status. In *Meeting proceedings of the IETF-93 Workshop, Prague*. Retrieved from https://www.ietf.org/proceedings/93/slides/slides-93-tls-8.pdf, 2015.

[49] B. Schmidt. Formal analysis of key exchange protocols and physical protocols, 2012. Doctoral thesis, ETH Zurich, Switzerland.

[50] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In S. Chong, editor, *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE, 2012.

[51] Tamarin prover GitHub repository (develop branch). https://github.com/tamarin-prover/tamarin-prover, 2015.

[52] Transport Layer Security Charter, February 2014. https://datatracker.ietf.org/wg/tls/charter.

[53] S. Vaudenay. 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*, pages 534–546, 2002.

[54] D. Wagner and B. Schneier. Analysis of the SSL 3.0 Protocol. In *In Proceedings of the Second UNIX Workshop on Electronic Commerce*, pages 29–40. USENIX Association, 1996.