A Comprehensive Symbolic Analysis of TLS 1.3

Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, Thyla van der Merwe

March 7, 2019

Baiyu Li

UC San Diego

Highlights

UC San Diego

Symbolic model and (semi) automated analysis of TLS 1.3

- The analysis covers *all* handshake modes of TLS 1.3
- Modular, flexible: OK with removal of 0-RTT
- Prove the majority of security requirements of TLS 1.3
  - Secrecy of session keys
  - Perfect forward secrecy of session keys
  - Peer authentication
  - Key compromise impersonation resistance
  - Key agreement and uniqueness across handshakes
- Uncover security problems in applications that assumes TLS 1.3 has strong auth guarantees
- Exhibiting the relation between specification and model: annotated specification

TLS 1.3 review

UC San Diego

- TLS 1.3 specifies three key-exchange modes: DHE, PSK, and PSK coupled with DHE
- Three post-handshake mechanisms covering traffic key updates, post-handshake client authentication, and sending of new session tickets (NST)
- Handshake protocol maintains a rolling transcript = hash(all handshake messages)

A *comprehensive* analysis must consider *interactions* between

- KE modes
- handshake variants
- post-handshake mechanisms

A good complement of the computation security proofs, as the symbolic model can be more easily formalized and machine checked.

Components in the analysis

UC San Diego

- DHE: Ephemeral DH keys
- PSK: No PFS
- PSK with DHE: Has PFS with limited number of PKC operations
- Group renegotiation: HelloRetryRequest
- NST: Binds identity to a resumption-specific secret, can be used by client as PSK
- PSK binder: Binds the PSK to the handshake
- Session resumption and PSK: Use a OOB key for a new session or an NST to resume the session
- 0-RTT: Application must provide its own replay protection at the application layer
- Post-handshake client authentication:
  - Server sends a CertificateRequest
  - Client cannot be sure about its auth status
- Key update
- Key derivation: Two secret inputs: DHE and PSK

Security goals and properties

UC San Diego

The handshake protocol is intended to negotiate crypto keys via Authenticated Key Exchange.

- independent keys to protect handshake messages and app data messages

A list of eight properties of the handshake protocol:

1. Establish the same session keys
2. Secrecy of the seccion keys
3. Peer authentication
4. Uniqueness of session keys
5. Downgrade protection (not modeled)
6. PFS
7. KCI resistance
8. Protection of endpoint identities

The Tamarin prover with a symbolic model

Tamarin is a tool to analyze security protocols in a symbolic model, with Dolev-Yao-style active attackers

- Assume perfect crypto: An adversary learns an encrypted message only if he knows the secret key
- Messages and operations are abstracted using terms
- A protocol is modeled as a labeled transition systems with states, where state transitions are defined using rules with actions
- Properties are specified using equational theories and guarded first order logic formulas. These can be either trace properties or observational equivalence properties
- An imaginary active attacker who has complete control over the network and can replay, insert, delay, delete and modify. Attacker can only perform actions defined in the rules

The Tamarin prover

Let's look at a simple protocol:

$$\begin{array}{ll} C \rightarrow S: & \mathsf{aenc(k, pkS)} \\ C \leftarrow S: & \mathsf{h(k)} \end{array}$$

The Tamarin prover

Let's look at a simple protocol:

$C \rightarrow S$: aenc(k, pkS)
$C \leftarrow S$: h(k)

- Specify this protocol
- Express its properties

The Tamarin prover

Let's look at a simple protocol:

$C \rightarrow S$: aenc(k, pkS)
$C \leftarrow S$: h(k)

- Specify this protocol
- Express its properties

```
theory SimpleProtocol
begin

builtins: hashing
functions: senc/2, sdec/2
equations:
   sdec(senc(x.1, x.2), x.2) = x.1

axiom one_ltk:
   "All A x y #i #j.
      ((GenLtk( A, x ) @ #i) &
       (GenLtk( A, y ) @ #j))
      ==> (#i = #j)"

rule ...
lemma ...
end
```

The Tamarin prover

Rules are used to specify the protocol: Consider $C \rightarrow S$: aenc(k, pkS)

```
Fr(~k)
!Pk($S, pkS)
```

$\longrightarrow \square \longrightarrow$

```
Client_1( $S, ~k )
Out( aenc(~k, pkS) )
```

The Tamarin prover

UC San Diego

Rules are used to specify the protocol: Consider $C \rightarrow S$: aenc(k, pkS)

```
Fr(~k)                 Client_1( $S, ~k )
!Pk($S, pkS)   --[ ]-->   Out( aenc(~k, pkS) )
```

- Terms model messages and identities:
  - Constants: `'c'`
  - Variables: `pkS`
  - Function applications: `f(t1,...,tn)`
- A constant or a variable may have a "sort" expressed with a prefix:
  - Fresh names: `~k`
  - Public names: `$A`
  - Temporal names: `#i`
  - Messages: `m`

The Tamarin prover

Rules are used to specify the protocol: Consider $C \rightarrow S$: aenc(k, pkS)

```
Fr(~k)
!Pk($S, pkS)
```

$\longrightarrow \square \longrightarrow$

```
Client_1( $S, ~k )
Out( aenc(~k, pkS) )
```

- Facts are built from terms: `Fr(~k)`, `!Pk($S, pkS)`
- A fact can be *linear* or *persistent*
  - A linear fact can be consumed only once
  - A persistent fact can be consumed multiple times
- Some special facts:
  - `Fr(x)`: `x` is a fresh name and it is freshly generated
  - `In(x)`: `x` is an incoming message from the network
  - `Out(x)`: `x` is an output message to the network
  - `K(x)`: `x` is known to the adversary

The Tamarin prover

Rules are used to specify the protocol: Consider $C \rightarrow S$: aenc(k, pkS)

```
Fr(~k)                --[ ]-->   Client_1( $S, ~k )
!Pk($S, pkS)                     Out( aenc(~k, pkS) )
```

- A rule has a name and three parts (sequences of facts):
  - Premise
  - Action
  - Conclusion
- A rule defines how the system can transit from one state to another
  - Facts in premise are consumed from the system state
  - Facts in conclusion are added to the system state
  - Facts in action are appended to *trace*

The Tamarin prover

UC San Diego

Rules are used to specify the protocol: Consider $C \to S$: aenc(k, pkS)

```
Fr(~k)                            Client_1( $S, ~k )
!Pk($S, pkS)        --> □ -->     Out( aenc(~k, pkS) )
```

This is written as:

```
rule Client_1:
    [ Fr(~k), !Pk($S, pkS) ]
  -->
    [ Client_1( $S, ~k ), Out( aenc(~k, pkS) ) ]
```

The Tamarin prover

Rules are used to specify the protocol: Consider $C \rightarrow S$: aenc(k, pkS)

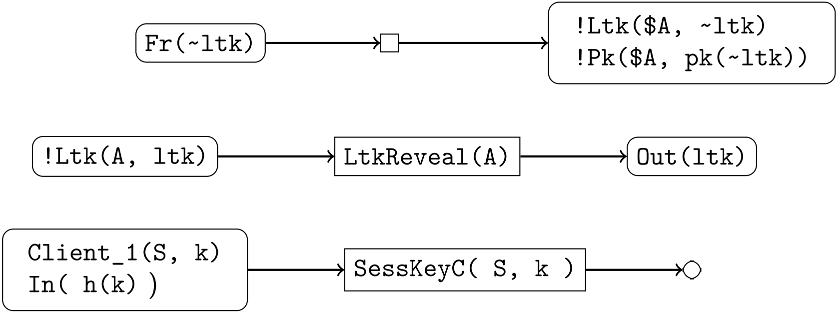
```
Fr(~k)                  Client_1( $S, ~k )
!Pk($S, pkS)   --[ ]-->  Out( aenc(~k, pkS) )
```

Tamarin provides some builtin function symbols and theories

- `hashing`: A function symbol `h/1` and no equations
- `asymmetric-encryption`: Two function symbols `aenc/2` and `adec/2`

  `adec(aenc(m, pk(sk)), sk) = m`

A few more rules...



The Tamarin prover

Now to want to express properties about the system and prove them.

- Trace properties are used to express system behaviours
  - Using first-order logic with a sort for timepoints
  - Formulas are guarded using $\exists$ and $\forall$
  - Usual logic operators `==>`, `&`, `|`, `not`
  - `f @ i` expresses an action constraint at timepoint `#i`
  - Timepoints are ordered: can assert `i < j` and `#i = j`!
  - Message terms can be compared with equality: `x = y`

The Tamarin prover

Now to want to express properties about the system and prove them.

- Trace properties are used to express system behaviours
  - Using first-order logic with a sort for timepoints
  - Formulas are guarded using $\exists$ and $\forall$
  - Usual logic operators `==>`, `&`, `|`, `not`
  - `f @ i` expresses an action constraint at timepoint `#i`
  - Timepoints are ordered: can assert `i < j` and `#i = j`!
  - Message terms can be compared with equality: `x = y`
- Observational equivalence is also possible, but with limited capability

The Tamarin prover

Now to want to express properties about the system and prove them.

For example:

```
lemma Client_session_key_secrecy2:
  "
     All S k #i #j.
       /* client has set up a session key 'k' with a server'S' */
     ( SessKeyC(S, k) @ #i
       /* and the adversary knows 'k' */
       & K(k) @ #j
     ) ==>
       /* Then K must have performed a long-term key reveal on 'S'. */
     Ex #r. LtkReveal(S) @ r
  "
```

The Tamarin prover

Now to want to express properties about the system and prove them.

For example:

```
lemma Client_session_key_secrecy2:
  "
     All S k #i #j.
       /* client has set up a session key 'k' with a server'S' */
     ( SessKeyC(S, k) @ #i
       /* and the adversary knows 'k' */
       & K(k) @ #j
     ) ==>
       /* Then K must have performed a long-term key reveal on 'S'. */
     Ex #r. LtkReveal(S) @ r
  "
```

To prove a lemma, we can let Tamarin try to prove it automatically, or by giving instructions on how to solve the constraint problem.

A comprehensive model

- A comprehensive symbolic model of TLS 1.3 is devied in the Tamarin's framework
- This covers all the possible interactions between each property
- The model closely matches the specification: an annotated TLS 1.3 specification

Thread model and security properties

UC San Diego

- An extension of the Dolve-Yao symbolic attacker
- The attacker can compromise
  - Long-term keys of parties
  - Pre-shared keys, whether created OOB or through NST
  - DH values

Thread model and security properties

Recall that there are eight required security properties:

1. Establish the same session keys
2. Secrecy of the seccion keys
3. Peer authentication
4. Uniqueness of session keys
5. Downgrade protection (not modeled)
6. PFS
7. KCI resistance
8. Protection of endpoint identities (not modeled)

Analysis results

UC San Diego

In general, the TLS 1.3 specification meets the required properties.

- A client and a server agrees on the secret session keys
- Session keys are unique across and within handshake instances
- PFS of session keys holds in suitable situations
- Authentication guarantees are satisfied in general

Analysis results

UC San Diego

In general, the TLS 1.3 specification meets the required properties.

- A client and a server agrees on the secret session keys
- Session keys are unique across and within handshake instances
- PFS of session keys holds in suitable situations
- Authentication guarantees are satisfied in general

Proofs are complicated...

- Multi-stage process: To break down the protocol models, sublemmas are used to give hints to the Tamarin prover
- Uses heuristics to help the automated prover to quickly re-proving large sections
- Still many manual proof efforts
- Model takes 10GB RAM, and takes about 100GB to compute the proof

A possible mismatch between client and server view

Some strong post-authentication guarantees are not met:

- Post-handshake authentication: Both parties should share a common view of the session
- The analysis shows that: when the server asks for a post-handshake client authentication, and the client responds, the client cannot be sure whether the server considers the channel is mutually authenticated
- TLS 1.3 working group has decided to let the application level handle this confirmation

The end!