Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate

Bruno Blanchet

INRIA Paris
Bruno.Blanchet@inria.fr

Joint work with Karthikeyan Bhargavan and Nadim Kobeissi

Year 2021-22
Transport Layer Security (TLS) 1.3

- Next version of the most popular secure channel protocol.
  - Completely redesigned from TLS 1.2
  - After 21 drafts, on the verge of standardization
Transport Layer Security (TLS) 1.3

- Next version of the most popular secure channel protocol.
  - Completely redesigned from TLS 1.2
  - After 21 drafts, on the verge of standardization
- Why did we need a new protocol?
  - Security: remove broken legacy crypto constructions
## Attacks against TLS 1.2

<table>
<thead>
<tr>
<th>Attack</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC4</td>
<td>Keystream biases</td>
<td>Mar’13</td>
</tr>
<tr>
<td>Lucky13</td>
<td>MAC-Encode-Encrypt CBC</td>
<td>Mar’13</td>
</tr>
<tr>
<td>POODLE</td>
<td>SSLv3 MAC-Encode-Encrypt CBC</td>
<td>Dec’14</td>
</tr>
<tr>
<td>FREAK</td>
<td>Export-grade 512-bit RSA</td>
<td>Mar’15</td>
</tr>
<tr>
<td>LOGJAM</td>
<td>Export-grade 512-bit DH</td>
<td>May’15</td>
</tr>
<tr>
<td>SLOTH</td>
<td>RSA-MD5 signatures</td>
<td>Jan’16</td>
</tr>
<tr>
<td>DROWN</td>
<td>SSLv2 PSA-PKCS#1v1.5 Enc</td>
<td>Mar’16</td>
</tr>
<tr>
<td>SWEET32</td>
<td>3DES Encryption</td>
<td>Oct’16</td>
</tr>
</tbody>
</table>
Transport Layer Security (TLS) 1.3

- Next version of the most popular secure channel protocol.
  - Completely redesigned from TLS 1.2
  - After 20 drafts, on the verge of standardization
- Why did we need a new protocol?
  - Security: remove broken legacy crypto constructions
  - Efficiency: reduce handshake roundtrip latency
    - 0-RTT when the client and server have a pre-shared key
    - 0.5-RTT

These are potentially contradictory goals
Needs extensive security analysis before deployment!
Transport Layer Security (TLS) 1.3

- Next version of the most popular secure channel protocol.
  - Completely redesigned from TLS 1.2
  - After 20 drafts, on the verge of standardization

- Why did we need a new protocol?
  - Security: remove broken legacy crypto constructions
  - Efficiency: reduce handshake roundtrip latency
    - 0-RTT when the client and server have a pre-shared key
    - 0.5-RTT
  - These are potentially contradictory goals

- Needs extensive security analysis before deployment!
  - The IETF called for academics to formally analyze the protocol drafts.
Analyzing TLS 1.3

Many published analyses for intermediate TLS 1.3 drafts

- Cryptography proofs (of drafts 5,9,10)
- Symbolic protocol analysis (of draft 10)
  [Cremers et al. S&P’16]
- Verified implementation (of draft 18 record protocol)
  [Bhargavan et al. S&P’17]
- Symbolic and computational proofs (of draft 18)
  [Bhargavan et al. S&P’17; this talk]

Are we done? Is it secure?

- If we deploy TLS 1.3, will it expose new attacks?
TLS 1.2 and its proofs: a checkered history

Historically, published proofs of TLS missed many attacks
Large gaps between simplified models and the deployed protocol

1. Proofs ignored “ugly” implementation details
   - e.g. AES-CBC padding, RSA-PKCS#1v1.5 padding

2. Proofs relied on strong crypto assumptions on primitives
   - e.g. collision resistant hash functions, strong Diffie-Hellman groups

3. Proofs ignored composition with obsolete/unpopular modes
   - e.g. SSLv2, EXPORT ciphers, renegotiation

How do we ensure that TLS 1.3 proofs do not fall into these traps?
Our approach

- Use automated verification tools to handle protocol complexity
  - Easy to extend as protocol evolves, or as we model new features
- Symbolically analyze protocol against known attack vectors
  - Find or prove the absence of downgrade attacks to TLS 1.2 (using ProVerif)
- Build a mechanically-checked cryptographic proof of TLS 1.3
  - Explore the crypto assumptions needed by TLS 1.3 (using CryptoVerif)
- Synchronize verified models with RFC and its implementations
  - Extract ProVerif model from an interoperable implementation (RefTLS)
Our vision: one model, three tasks

Protocol fix → TLS 1.3 Model → Potential attack

TLS 1.3 Model

- ProVerif
  - Symbolic proof

- CryptoVerif
  - Cryptographic proof

Reference implementation

Other TLS libraries

Interop testing

(Inspired by: Verified interoperable implementations of security protocols, TOPLAS 2008.)
Our current toolchain

- Protocol fix
  - TLS 1.3 Core protocol code
    - Model extraction
      - TLS 1.3 Symbolic model
        - ProVerif
          - Symbolic proof
      - TLS 1.3 Crypto model
        - CryptoVerif
          - Cryptographic proof
    - Manual edits
  - Potential attack
  - Reference implementation
  - Other TLS libraries
  - Interop testing
Symbolic analysis to find downgrade (and other) attacks

Recent attacks on legacy crypto in TLS:

<table>
<thead>
<tr>
<th>Attack</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC4</td>
<td>Keystream biases</td>
<td>[Mar’13]</td>
</tr>
<tr>
<td>Lucky13</td>
<td>MAC-Encode-Encrypt CBC</td>
<td>[Mar’13]</td>
</tr>
<tr>
<td>POODLE</td>
<td>SSLv3 MAC-Encode-Encrypt</td>
<td>[Dec’14]</td>
</tr>
<tr>
<td>FREAK</td>
<td>Export-grade 512-bit RSA</td>
<td>[Mar’15]</td>
</tr>
<tr>
<td>LOGJAM</td>
<td>Export-grade 512-bit DH</td>
<td>[May’15]</td>
</tr>
<tr>
<td>SLOTH</td>
<td>RSA-MD5 signatures</td>
<td>[Jan’16]</td>
</tr>
<tr>
<td>DROWN</td>
<td>SSLv2 PSA-PKCS#1v1.5 Enc</td>
<td>[Mar’16]</td>
</tr>
</tbody>
</table>

Legacy crypto remains in TLS libraries for backwards compatibility.
Is TLS 1.3 secure, if it is deployed alongside older versions of TLS?

- Can a man-in-the-middle **downgrade** TLS 1.3 peers to use legacy crypto?
Modeling weak crypto in ProVerif

- Classic symbolic (Dolev-Yao) protocol models idealize crypto
  - Perfect black-boxes that cannot be opened without relevant key
- We model agile crypto primitives parameterized by algorithm
  - Given a **strong** algorithm, the primitive behaves ideally
  - Given a **weak** algorithm, the primitive completely breaks
Modeling weak crypto in ProVerif

- Classic symbolic (Dolev-Yao) protocol models idealize crypto
  - Perfect black-boxes that cannot be opened without relevant key
- We model agile crypto primitives parameterized by algorithm
  - Given a strong algorithm, the primitive behaves ideally
  - Given a weak algorithm, the primitive completely breaks
  - e.g. a weak Diffie-Hellman group behaves like a trivial 1-element group

```plaintext
fun dhIdeal(element, bitstring): element.
equation forall x: bitstring, y: bitstring;
   dhIdeal(dhIdeal(G, x), y) = dhIdeal(dhIdeal(G, y), x).

fun dhExp(group, element, bitstring): element
reduc forall g: group, e: element, x: bitstring;
   dhExp(WeakDH, e, x) = BadElement
otherwise forall g: group, e: element, x: bitstring;
   dhExp(StrongDH, BadElement, x) = BadElement
otherwise forall g: group, e: element, x: bitstring;
   dhExp(StrongDH, e, x) = dhIdeal(e, x).
```
Modeling weak crypto in ProVerif

- Classic symbolic (Dolev-Yao) protocol models idealize crypto
  - Perfect black-boxes that cannot be opened without relevant key
- We model agile crypto primitives parameterized by algorithm
  - Given a **strong** algorithm, the primitive behaves ideally
  - Given a **weak** algorithm, the primitive completely breaks
  - e.g. a weak Diffie-Hellman group behaves like a trivial 1-element group
  - Similarly, we model strong and weak authenticated encryption, hash functions, MACs, RSA encryption and signatures.
- **Our model is overly conservative, it may not indicate real exploits**
  - Our goal is to verify TLS 1.3 against future attacks on legacy crypto
Modeling TLS 1.3 in ProVerif

TLS 1.3 1-RTT handshake
- 12 messages in 3 flights, 16 derived keys, then data exchange
+ PSK-based 0-RTT
+ TLS 1.2
+ Agile Crypto: ~400 lines
+ TLS models: ~500 lines

Modeling is easy, verification takes effort

Key Derivation Functions:
HKDF-Extract\(k, s\) = HMAC-H^k(s)
\(\text{hkdf-expand-label}_1(l, h) = \text{HMAC-H}^\{\text{len}_l\}||\text{"TLS 1.3," ||}||h||0x01\)
\(\text{Derive-Secret}(l, m) = \text{hkdf-expand-label}_1(l, H(m))\)

1-RTT Key Schedule:
\(\text{kdf}_0 = \text{HKDF-Extract}(0, \text{len}_0)\)
\(\text{kdf}_s = \text{HKDF-Extract}(\text{es}, \text{em}_s)\)
\(\text{kdf}_s = \text{HKDF-Extract}(\text{es}, \text{em}_s)\)
\(k_c = \text{hkdf-expand-label}(\text{h}_c, \text{key}, \text{""})\)
\(k_m = \text{hkdf-expand-label}(\text{h}_c, \text{finished}, \text{""})\)
\(k_s = \text{hkdf-expand-label}(\text{h}_s, \text{key}, \text{""})\)
\(k_m = \text{hkdf-expand-label}(\text{h}_s, \text{finished}, \text{""})\)
\(\text{kdf}_s(\text{ms}, \text{log}_4) = k_c, k_s, \text{ems} \text{where}\)
\(\text{ats}_c = \text{Derive-Secret}(\text{ms}, \text{ats}_c, \text{log}_4)\)
\(\text{ats}_s = \text{Derive-Secret}(\text{ms}, \text{ats}_s, \text{log}_4)\)
\(\text{ems} = \text{Derive-Secret}(\text{ms}, \text{ems}, \text{log}_4)\)
\(k_c = \text{hkdf-expand-label}(\text{ats}_c, \text{key}, \text{""})\)
\(k_s = \text{hkdf-expand-label}(\text{ats}_s, \text{key}, \text{""})\)
\(\text{kdf}_s(\text{ms}, \text{log}_7) = \text{psk}' \text{ where}\)
\(\text{PSK-based Key Schedule:}\)
\(\text{kdf}_s(\text{psk}) = \text{es}, k^b\) where
\(\text{es} = \text{HKDF-Extract}(0, \text{len}_{\text{psk}})\)
\(k^b = \text{Derive-Secret}(\text{es}, \text{pbk}, \text{""})\)
\(\text{kdf}_\text{0-RTT}(\text{es}, \text{log}_1) = k_c\) where
\(\text{ets}_c = \text{Derive-Secret}(\text{es}, \text{ets}_c, \text{log}_1)\)
\(k_c = \text{hkdf-expand-label}(\text{ets}_c, \text{key}, \text{""})\)
Writing and verifying security goals

- **We state security queries for data sent between honest users**
  - **Secrecy**: messages between honest peers are unknown to an adversary
  - **Authenticity**: messages between honest peers cannot be tampered
  - **Replay prevention**: messages between honest peers cannot be replayed
  - **Forward secrecy**: secrecy holds even if the peers’ long-term keys are leaked after the session is complete

- **Secrecy query for \( \text{msg}(conn, S) \) sent from anonymous \( C \) to server \( S \)**

\[
\text{query} \ \text{attacker}(\text{msg}(conn, S)) \implies \text{false}
\]
Refining security queries

 QUERY: is msg\((conn, S)\) secret?

\textbf{query} \ attacker(msg\((conn, S)\)) \implies \mathrm{false}

 FALSE: ProVerif finds a counterexample if S’s private key is compromised.
Refining security queries

- QUERY: is \( \text{msg} (\text{conn}, S) \) secret as long as \( S \) is uncompromised?

\[
\text{query} \ \text{attacker} (\text{msg} (\text{conn}, S)) \implies \\
\text{event} (\text{WeakOrCompromisedKey} (S))
\]

- FALSE: ProVerif finds a counterexample if the AE algorithm is weak.
Refining security queries

 QUERY: Strongest secrecy query that can be proved in our model

query attacker(msg(conn, S)) ⇒
 event(WeakOrCompromisedKey(S)) ∨
 event(ServerChoosesAE(conn, S, WeakAE)) ∨
 event(ServerChoosesKEX(conn, S, WeakDH)) ∨
 event(ServerChoosesKEX(conn′, S, WeakRSAEncryption)) ∨
 event(ServerChoosesHash(conn′, S, WeakHash))

 TRUE: ProVerif finds no counterexample
Conclusion: Downgrade security for TLS 1.2 + TLS 1.3

- Messages on a TLS 1.3 connection between honest peers are secret:
  1. if the connection does not use a weak AE algorithm,
  2. the connection does not use a weak DH group,
  3. the server never uses a weak hash algorithm for signing, and
  4. the server never participates in a TLS 1.2 RSA key exchange.

- Analysis confirms preconditions for downgrade resilience in TLS 1.3
  - identifies weak algorithms in TLS 1.2 that can harm TLS 1.3 security
Mechanized computational proof

- **Mechanized verification of TLS 1.3 Draft-18 in the computational model.**
  - + Handshake with PSK and/or DHE.
  - + Handshake with and without client authentication.
  - + 0-RTT and 0.5-RTT data, key updates.
  - – No post-handshake authentication.
  - – No version or ciphersuite negotiation: only strong algorithms.
  - – For PSK-DHE, we do not prove forward secrecy wrt. the compromise of PSK.

- We prove security properties of the initial handshake, the handshake with pre-shared key, and the record protocol using CryptoVerif.

- We compose these pieces manually.
Structure of the proof

1. Computational assumptions
2. Lemmas on primitives
3. Protocol pieces
   - Handshake without pre-shared key
   - Handshake with pre-shared key (PSK and PSK-DHE)
   - Record protocol
4. Compose the pieces together
Structure of the proof: final composition

Handshake without pre-shared key

Handshake with pre-shared key

Record protocol

updated ts

ats_c

ats_s

psk'

ets_c

Bruno Blanchet (INRIA)
Key schedule (Draft-18, excerpt)

PSK \xrightarrow{HKDF-Extract} 0
\quad \text{Early Secret es}
\quad \text{Derive-Secret(., “external psk binder key” | “resumption psk binder key”, “”) = binder_key}
\quad \text{Derive-Secret(., “client early traffic secret”, ClientHello) = client_early_traffic_secret (ets}_c)
Assumptions (1)

- **Diffie-Hellman:**
  - Gap Diffie-Hellman (GDH)
    - needed in particular for 0.5-RTT.
  - The probability that \( g^x = Y \) where \( x \) is random and \( Y \) is independent of \( x \) is negligible.
  - The probability that \( g^{x\cdot y} = Y \) where \( x \) and \( y \) are independent random private keys and \( Y \) is independent of \( x \) or \( y \) is negligible.
    - true both for prime-order groups and for Curve25519.
  - Diffie-Hellman group elements different from \( 0^{\text{len}_H()} \)
    - avoids confusion between handshakes with and without Diffie-Hellman exchange.
  - Diffie-Hellman group elements different from \( \text{len}_H() \| \text{"TLS 1.3," } \| l \| h \| 0x01 \)
    - avoids collision between HKDF-Extract(es, e) and Derive-Secret(es, pbk, "") or Derive-Secret(es, ets_c, log_1).
    - independently discovered and discussed on the TLS mailing list.
    - change in Draft-19 makes this assumption unnecessary: add a Derive-Secret stage before HKDF-Extract.
Assumptions (2)

- **Signatures**: sign is UF-CMA.
- **Hash functions**: H is collision-resistant.
- **HMAC**:
  - $x \mapsto \text{HMAC}-H^{0_{\text{len}_H}}(x)$ and $x \mapsto \text{HMAC}-H^{k_{\text{df}_0}}(x)$ are independent random oracles.
  - HMAC-H is a PRF, for keys different from $0^{\text{len}_H}$ and $k_{\text{df}_0}$.
- **Authenticated Encryption**: IND-CPA and INT-CTXT provided the same nonce is never used twice with the same key.
Lemmas on primitives: MAC and signatures

- $\text{mac}_k^H(m) = \text{mac}_k^H(H(m))$ is an SUF-CMA MAC.
- $\text{sign}_{sk_H}^m(m) = \text{sign}_{sk}^H(H(m))$ is an UF-CMA signature.
Lemmas on primitives: key schedule

Lemma

When $es$ is a fresh random value,

- $e \mapsto \text{HKDF-Extract}(es, e)$ and
- $log_1 \mapsto \text{Derive-Secret}(es, ets_c, log_1)$

are indistinguishable from independent random functions, and

- $k^b = \text{Derive-Secret}(es, pbk, "")$ and
- $\text{HKDF-Extract}(es, 0^{len_H()})$

are indistinguishable from independent fresh random values independent from these random functions.

- Proved using CryptoVerif.
- Similar lemmas for other parts of the key schedule.
- Used as assumption in the proof of the protocol.
Handshake without pre-shared key: model

- Model a honest client and a honest server.
- May interact with dishonest clients and servers included in the adversary.
- Ignore negotiation (RetryRequest).
- Give the handshake keys to adversary:
  - The adversary can encrypt and decrypt messages.
  - The security proof does not rely on that.
- Server always authenticated.
- With and without client authentication.
- The honest client and server may be dynamically compromised.
Handshake without pre-shared key: honest sessions

- The **client** is in a **honest session** if
  - the server public key is the one of the honest server, and
  - the honest server is not compromised, or it is compromised and the messages received by the client have been sent by the honest server.

- The **server** is in a **honest session** if
  - client authenticated:
    - the client public key is the one of honest client, and
    - the honest client is not compromised, or it is compromised and the messages received by the server have been sent by the honest client.
  - client not authenticated: the Diffie-Hellman share received by the server has been sent by the honest client.
Handshake without pre-shared key: security (1)

- **Key authentication:**
  - If the honest client terminates a honest session, then the honest server has accepted a session with that client, and they agree on:
    - keys $ats_c$, $ats_s$, and $ems$,
    - all messages until the server $Finished$ message.
  - If the honest server terminates a honest session, then the honest client has accepted a session with that server, and they agree on the keys and on all messages.

- **Replay prevention:** the previous properties are injective.

- **Key secrecy:** the keys
  - $ats_c$, $ems$, $psk'$ client side, when the client terminates a honest session;
  - $ats_s$ server side, when the server sends its $Finished$ message and the received Diffie-Hellman share comes from the client (for 0.5-RTT) are indistinguishable from independent fresh random values.
Handshake without pre-shared key: security (2)

- **Same key:**
  - If the honest client terminates a honest session and the honest server has accepted a session with the same messages, then they have the same key.
  - If the honest server terminates a honest session and the honest client has accepted a session with the same messages, then they have the same key.

- **Unique channel identifier:**
  - $psk'$ or $H(log_7)$:
    If a client session and a server session have the same $psk'$ or $H(log_7)$, then all their parameters are equal (collision-resistance).
  - $ems$:
    If a client session and a server session have the same $ems$, then they have the same $log_4$ (collision-resistance), so all their parameters are equal (CryptoVerif).
Handshake without pre-shared key: guidance

- Signature under $sk_S$.
- Introduce tests to distinguish cases, depending on
  - whether the Diffie-Hellman share received by the server is a share $g^{x'}$ from the client,
  - and whether the Diffie-Hellman share received by the client is the share $g^y$ generated by the server upon receipt of $g^{x'}$.
- Random oracle assumption on $x \mapsto \text{HMAC-H}^{kdf_0}(x)$.
- Replace variables that contain $g^{x'y}$ with their values to make equality tests $m = g^{x'y}$ appear.
- Gap Diffie-Hellman assumption.
  - $\Rightarrow$ the handshake secret $hs$ is a fresh random value.
  - Lemmas on key schedule $\Rightarrow$ other keys are fresh random values.
- MAC.
- Signature under $sk_C$. 
Handshake with pre-shared key: model

- Includes handshakes with and without Diffie-Hellman exchange.
- Includes 0-RTT.
- Ignore the ticket $\text{enc}^{k_t}(psk)$; consider a honest client and a honest server that share the PSK.
- Give the handshake keys to adversary (as before).
- Certificates optional, since the client and server are already authenticated by the PSK.
Handshake with pre-shared key: security (1)

Same properties as for the initial handshake, but

- **No compromise of PSK.**
  - Limitation of CryptoVerif: cannot prove forward secrecy wrt. to the compromise of PSK for PSK-DHE.

- **Weaker properties for 0-RTT:**
  - **Key authentication:** No authentication for $ets_c$:
    - several binders, and only one of them is checked;
    - the adversary can alter the others, yielding a different $ets_c$ server-side.
  - **Replay prevention:** No replay protection for $ets_c$.
  - **Secrecy of keys:** The keys $ets_c$ server-side are not independent of each other, due to the replay.
Handshake with pre-shared key: security (2)

For 0-RTT, we show:

- **Client-side:** The keys $ets_c$ are indistinguishable from independent random values.
- **Server-side:**
  - If the received ClientHello message has been sent by the client, then this session matches a session of the client with same key $ets_c$.
  - Otherwise,
    - If the ClientHello message has been received before, then the key $ets_c$ computed by the server is the same as in the previous session with the same ClientHello message.
    - Otherwise, the key $ets_c$ computed by the server is indistinguishable from a fresh random value, independent from other keys.
Record protocol

The client and the server share a fresh random traffic secret.

- **Key secrecy**: The updated traffic secret is indistinguishable from a fresh random value.

- **Message secrecy**: When the adversary provides two sets of plaintexts $m_i$ and $m'_i$ of the same padded length, it is unable to determine which set is encrypted, even when the updated traffic secret is leaked.

- **Message Authentication**: If a message $m$ is decrypted by the receiver with a counter $c$, then the message $m$ has been encrypted and sent by an honest sender with the same counter $c$.

- **Replay Prevention**: The authentication property above is injective.
Composition

Handshake without pre-shared key

Handshake with pre-shared key

Record protocol

updated $ts$

$ats_c$, $ats_s$, $psk'$, $ets_c$
Composition: main theorem (informal)

- System $S$: key exchange; $A$ and $B$ obtain a key such that:
  - **Key secrecy**: The keys obtained by $A$ are indistinguishable from independent random values.
  - **One-way injective authentication**: For each session of $B$ that obtains a key $k$ with session identifier $sid$, there is a distinct session of $A$ that obtains the key $k$ with session identifier $sid$.
  - **Uniqueness**: There is a single session of $A$ with a given session identifier $sid$.
- System $S'$ assumes a fresh random key shared by $A'$ and $B'$.
- The composed system $S_{\text{composed}}$ runs the key exchange followed by $A'$ with the key obtained by $A$ and $B'$ with the key obtained by $B$.
- The security properties of $S$ and $S'$ carry over to $S_{\text{composed}}$. 
The previous theorem allows to perform most compositions.
More tricky composition theorems for 0-RTT, because the properties are weaker.
A simpler composition theorem for key update.
Mechanized computational proof: conclusion

- Mechanized verification of TLS 1.3 Draft-18 in the computational model.
  - Handshake with PSK and/or DHE.
  - Handshake with and without client authentication.
  - 0-RTT and 0.5-RTT data, key updates.
  - No post-handshake authentication.
  - No version or ciphersuite negotiation: only strong algorithms.
  - For PSK-DHE, we do not prove forward secrecy wrt. the compromise of PSK.

- CryptoVerif proves properties of the handshake with (resp. without) pre-shared-key and of the record protocol.

- We infer properties of the whole system by manual composition.

- Modular approach essential to be able to handle such a complex protocol.

- TLS 1.3 Draft-18 is well-designed to allow such a proof.
RefRLS: a reference implementation

- Supports TLS 1.0-1.3 and interoperates with other libraries
  - Supports Draft 20 1-RTT with (EC)DHE and/or PSK (No 0-RTT)
  - Supports common TLS 1.2 modes (RSA, DHE with AES-CBC, AES-GCM)
- Distributed as a JavaScript library for ease of deployment
  - Can be used within Node.js and Electron apps
  - Meant for early adopters and interop testing, not for production code!
- We extract core protocol functions from the implementation
  - Ensures that we did not miss some RFC/implementaiton details
  - Other parts of the implementation are not verified (unlike miTLS)
RefTLS architecture

Mostly written in Flow

- Statically-typed JavaScript
- Identify, isolate protocol core
- Protocol state machine
- Includes all crypto processing: encryption, signing, DHE, ...

Core written in ProScript

- Typed JavaScript subset that can be compiled to ProVerif

[Kobeissi et al. EuroS&P’17]
Results and limitations

- We present a comprehensive analysis of TLS 1.3 draft 18
  - Symbolic analysis, cryptographic proofs, a reference implementation
- Many limitations, missing features, unverified components
  - Symbolic model ignores resumption, post-handshake authentication
  - Crypto proof ignores negotiation, legacy versions, post-handshake authentication
  - Unverified protocol code: message parsing, crypto library, Node

http://github.com/inria-prosecco/reftls