Composition Theorems for CryptoVerif and Application to TLS 1.3

Bruno Blanchet

INRIA Paris
Bruno.Blanchet@inria.fr

Year 2023-24
Introduction

- **Composition** between
  - a key exchange protocol
  - a protocol that uses the key

- Results stated in the CryptoVerif framework:
  - computational model
  - formal framework for stating the composition theorem
  - prove bigger protocols in CryptoVerif
  - prove protocols with loops in CryptoVerif

Adapt and extend previous computational composition results by Brzuska, Fischlin et al. [CCS’11, CCS’14 and CCS’15]
Application to TLS 1.3

Why TLS 1.3?

- Important protocol, just standardized
- Well designed to allow composition
- Contains loops:
  - Unbounded number of handshakes and key updates
- Variety of compositions:
  - In most cases, the key exchange provides injective authentication
  - For 0-RTT data = data sent by the client to the server immediately after the message (ClientHello):
    - possible replay, so non-injective authentication
    - variant for the case of altered ClientHello
  - Simpler composition theorem for key updates
Security properties proved by CryptoVerif

- **Indistinguishability**: $Q \approx^V Q'$ when an adversary with access to the variables $V$ has a negligible probability of distinguishing $Q$ from $Q'$.

- **Secrecy**: $Q$ preserves the secrecy of $x$ with public variables $V$ when an adversary with access to the variables $V$ has a negligible probability of distinguishing the values of $x$ in several sessions from independent random values.

- **Correspondences**: If some events have been executed, then other events have been executed. Example:

  $$\text{event}(e_1(x)) \rightarrow \text{event}(e_2(x))$$

  $Q$ satisfies the correspondence $corr$ with public variables $V$ when an adversary with access to the variables $V$ has a negligible probability of breaking $corr$. 
The most basic composition theorem
The most basic composition theorem

**Theorem (Assumptions)**

Let $C$ be any context with one hole, without replications above the hole. Let $M$ be a term of type $T$. Let

$$S_1 = C[\text{let } k = M \text{ in return}(); Q_1]$$

$$S_2 = O_2() := k \leftarrow T; \text{return}(); Q_2$$

where $k$ is the only variable common to $S_1$ and $S_2$; $S_1$ and $S_2$ have no common oracle, no common event, and no common table; $S_1$ and $S_2$ do not contain oracle calls; and $k$ does not occur in $C$ and $Q_1$.

Let

$$S_{\text{composed}} = C[\text{let } k = M \text{ in return}(); (Q_1 \mid Q_2)]$$
The most basic composition theorem

**Theorem (First conclusion)**

\[
S_1 = C[\texttt{let } k = M \texttt{ in return(); } Q_1]
\]
\[
S_2 = O_2() := k \overset{R}{\leftarrow} T; \texttt{return(); } Q_2
\]
\[
S_{\text{composed}} = C[\texttt{let } k = M \texttt{ in return(); } (Q_1 \mid Q_2)]
\]

1. If \( S_1 \) preserves the secrecy of \( k \) with public variables \( V \ (k \notin V) \), then we can transfer security properties from \( S_2 \) to \( S_{\text{composed}} \).

\( S_{\text{composed}} \) with the events of \( S_1 \) removed is indistinguishable with public variables \( V \cup (\text{var}(S_2) \setminus \{k\}) \) from an evaluation context interacting with \( S_2 \).

**Intuition:** The secrecy of \( k \) allows us to replace \( k \) with a random key.
The most basic composition theorem

**Theorem (Second conclusion)**

\[
\begin{align*}
S_1 &= C[\text{let } k = M \text{ in return}(); Q_1] \\
S_2 &= O_2() := k \leftarrow T; \text{return}(); Q_2 \\
S_{\text{composed}} &= C[\text{let } k = M \text{ in return}(); (Q_1 \mid Q_2)]
\end{align*}
\]

We can transfer security properties from \( S_1 \) to \( S_{\text{composed}} \), provided they are proved with public variable \( k \).

\( S_{\text{composed}} \) is indistinguishable with public variables \( \text{var}(S_{\text{composed}}) \) from an evaluation context interacting with \( S_1 \) with access to \( k \).
Main theorem

\[ S_1: \quad \begin{array}{c}
A \\
k_A \downarrow \\
B \\
k_B \uparrow
\end{array} \]

\[ S_2: \quad \begin{array}{c}
k \xleftarrow{R} T \\
A \downarrow \\
B \\
\end{array} \]

\[ S_{\text{composed}}: \quad \begin{array}{c}
A \\
k_A \downarrow \\
B \\
k_B \uparrow
\end{array} \]

\( S_1 \) may run several sessions of \( A \) and \( B \).
Replicating $S_2$

Consider:

$$S_2 = O() := \ldots O_1(y : T) := \ldots \text{event } e(M) \ldots$$

$$\text{insert } T(M') \ldots \text{get } T(z) \text{ such that} \ldots$$

We want to replicate $S_2$:

$$\text{foreach } \tilde{i} \leq \tilde{n} \text{ do } O() := \ldots O_1(y : T) := \ldots \text{event } e(M) \ldots$$

$$\text{insert } T(M') \ldots \text{get } T(z) \text{ such that} \ldots$$
Replicating $S_2$

Consider:

\[ S_2 = O() := \ldots O_1(y : T) := \ldots \text{event } e(M) \ldots \]

\[ \text{insert } T(M') \ldots \text{get } T(z) \text{ such that } \ldots \]

We want to replicate $S_2$:

\[ \text{foreach } \tilde{i} \leq \tilde{n} \text{ do } O[\tilde{i}()] := \ldots O_1[\tilde{i}](y[\tilde{i}] : T) := \ldots \text{event } e(M) \ldots \]

\[ \text{insert } T(M') \ldots \text{get } T(z[\tilde{i}]) \text{ such that } \ldots \]

Variables and oracles implicitly with indices of replication.
Replicating $S_2$

Consider:

\[
S_2 = O() := \ldots O_1(y : T) := \ldots \text{event } e(M) \ldots \\
\text{insert } T(M') \ldots \text{get } T(z) \text{ such that } \ldots
\]

We want to replicate $S_2$:

\[
\text{foreach } \tilde{i} \leq \tilde{n} \text{ do } O[\tilde{i}]() := \ldots O_1[\tilde{i}](y[\tilde{i}] : T) := \ldots \text{event } e(\tilde{i}, M) \ldots \\
\text{insert } T(\tilde{i}, M') \ldots \text{get } T(= \tilde{i}, z[\tilde{i}]) \text{ such that } \ldots
\]

We could add indices to events and tables to distinguish the various sessions.
Replicating $S_2$

Consider:

\[
S_2 = O() := \ldots O_1(y : T) := \ldots \text{event } e(M)\ldots \\
\text{insert } T(M') \ldots \text{get } T(z) \text{ such that } \ldots
\]

We want to replicate $S_2$:

\[
\text{foreach } \tilde{i} \leq \tilde{n} \text{ do } O[\tilde{i}]() := \ldots O_1[\tilde{i}](y[\tilde{i}] : T) := \ldots \text{event } e(\tilde{i}, M)\ldots \\
\text{insert } T(\tilde{i}, M') \ldots \text{get } T(= \tilde{i}, z[\tilde{i}]) \text{ such that } \ldots
\]

Problem: this is not preserved by composition.

In the key exchange, partenered sessions exchange the same messages, but may not have the same replication indices.

Also in the composed system.
Replicating $S_2$

Consider:

$$S_2 = O() := \ldots O_1(y : T) := \ldots \text{event } e(M) \ldots$$

$$\text{insert } T(M') \ldots \text{get } T(z) \text{ such that} \ldots$$

We want to replicate $S_2$:

$$\text{foreach } \tilde{i} \leq \tilde{n} \text{ do } O[\tilde{i}](x : T_{\text{sid}}) \ldots O_1[\tilde{i}](y[\tilde{i}] : T) := \ldots \text{event } e(x, M) \ldots$$

$$\text{insert } T(x, M') \ldots \text{get } T(= x, z[\tilde{i}]) \text{ such that} \ldots$$

Partnered sessions can be determined by a session identifier computed from the messages in the protocol.

The protocol that uses the key receives the session identifier in a variable $x$. 
Replicating $S_2$

Consider:

\[ S_2 = O() := P \]
\[ P = \ldots O_1(y : T) := \ldots \text{event } e(M) \ldots \]
\[ \quad \text{insert } T(M') \ldots \text{get } T(z) \text{ such that } \ldots \]

We replicate $S_2$:

\[ S_2! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, O', T_{\text{sid}}, S_2) \]
\[ = \text{foreach } \tilde{i} \leq \tilde{n} \text{ do } O'[\tilde{i}](x : T_{\text{sid}}) := \]
\[ \quad \text{if that value of } x \text{ never used before then} \]
\[ \text{AddIdxSid}(\tilde{i} \leq \tilde{n}, x : T_{\text{sid}}, P) \]
\[ \text{AddIdxSid}(\tilde{i} \leq \tilde{n}, x : T_{\text{sid}}, P) = \ldots O_1[\tilde{i}](y[\tilde{i}] : T) := \ldots \text{event } e(x, M) \ldots \]
\[ \quad \text{insert } T(x, M') \ldots \text{get } T(= x, z[\tilde{i}]) \text{ such that } \ldots \]

Never use the same session identifier twice.
Replicating $S_2$

Consider:

$$S_2 = O() := P$$

$$P = \ldots O_1(y : T) := \ldots \text{event } e(M) \ldots$$

$$\text{insert } T(M') \ldots \text{get } T(z) \text{ such that } \ldots$$

We replicate $S_2$:

$$S_2! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, O', T_{sid}, S_2)$$

$$= \text{foreach } \tilde{i} \leq \tilde{n} \text{ do } O'[\tilde{i}](x : T_{sid}) :=$$

$$\text{find } \tilde{u} = \tilde{i}' \leq \tilde{n} \text{ such that } \text{defined}(x[\tilde{i}'], x'[\tilde{i}'])$$

$$\land x = x[\tilde{i}'] \text{ then yield } \text{else }$$

$$\text{let } x' = \text{cst} \text{ in } \text{AddIdxSid}(\tilde{i} \leq \tilde{n}, x : T_{sid}, P)$$

$$\text{AddIdxSid}(\tilde{i} \leq \tilde{n}, x : T_{sid}, P) = \ldots O_1[\tilde{i}](y[\tilde{i}] : T) := \ldots \text{event } e(x, M) \ldots$$

$$\text{insert } T(x, M') \ldots \text{get } T(= x, z[\tilde{i}]) \text{ such that } \ldots$$
Replicating $S_2$: transfer of security properties

**Theorem**

Let $Q! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, c', T_{\text{sid}}, Q)$ and $Q'! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, c', T_{\text{sid}}, Q')$.

1. If $Q$ and $Q'$ do not contain events and $Q \approx^V Q'$, then $Q! \approx^V Q'!$.
2. If $Q$ preserves the secrecy of $y$ with public variables $V$, then so does $Q!$.
3. If $Q$ satisfies $\text{event}(e_1(y)) \Rightarrow \text{event}(e_2(y))$ with public variables $V$, then $Q!$ satisfies $\text{event}(e_1(x, y)) \Rightarrow \text{event}(e_2(x, y))$ with public variables $V$.

(Add a variable session identifier at the beginning of each event.)
Main composition theorem

\[ S_1: \quad S_{\text{composed}}: \]

\[ \begin{align*}
S_1: & \quad \text{AddReplMsg} \\
& \quad \begin{array}{c}
\text{A} \\
\quad \downarrow k_A \\
\text{B} \\
\quad \downarrow k_B \\
\text{A} & \quad \text{B}
\end{array}
\]

\[ S_{\text{composed}}: \\
\begin{array}{c}
\text{A} \\
\quad \downarrow k_A \\
\text{B} \\
\quad \downarrow k_B \\
\text{A} & \quad \text{B}
\end{array}
\]

\( (S_1 \text{ may run several sessions of } A \text{ and } B. ) \)
Main composition theorem

Theorem ($S_1$ and $S_2!$)

$$S_1 = C[\text{event } e_A(\text{sid}(\tilde{\text{msg}}_A), k_A, \tilde{i}); \text{let } k'_A = k_A \text{ in return}(M_A); Q_{1A}, \text{event } e_B(\text{sid}(\tilde{\text{msg}}_B), k_B); \text{return}(M_B); Q_{1B}]$$

$$S_2 = O_2() := k \xleftarrow{R} T; \text{return}(); (Q_{2A} \mid Q_{2B})$$

$$S_2! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, O'_2, T_{\text{sid}}, S_2)$$

where

1. $O'_2, k'_A, e_A, e_B$ do not occur elsewhere in $S_1, S_2!$;
2. $S_1$ and $S_2!$ have no common variable, oracle, event, table;
3. $S_1$ and $S_2!$ do not contain \texttt{newOracle} nor oracle calls;
4. and there is no \texttt{defined} condition in $S_2$. 

Main composition theorem

C is a context with two holes, with replications \( \text{foreach } \tilde{i} \leq \tilde{n} \text{ do } \) above the first hole and \( \text{foreach } \tilde{i}' \leq \tilde{n}' \text{ do } \) above the second hole

\[
S_1 = C[\text{event } e_A(\text{sid}(\tilde{\text{msg}}_A), k_A, \tilde{i}); \text{let } k'_A = k_A \text{ in return}(M_A); Q_{1A}, \text{event } e_B(\text{sid}(\tilde{\text{msg}}_B), k_B); \text{return}(M_B); Q_{1B}]
\]

\[
S_2 = O_2() := k \overset{R}{\leftarrow} T; \text{return}(); (Q_{2A} \mid Q_{2B})
\]

\[
S_{2!} = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, O'_2, T_{\text{sid}}, S_2)
\]

where

1. \( O'_2, k'_A, e_A, e_B \) do not occur elsewhere in \( S_1, S_{2!} \);
2. \( S_1 \) and \( S_{2!} \) have no common variable, oracle, event, table;
3. \( S_1 \) and \( S_{2!} \) do not contain \textbf{newOracle} nor oracle calls;
4. and there is no \textbf{defined} condition in \( S_2 \).
Main composition theorem

**Theorem (\(S_1\) and \(S_{2!}\))**

\[
S_1 = C[\textbf{event } e_A(\text{sid}(\widetilde{msg}_A), k_A, \widetilde{i}); \textbf{let } k'_A = k_A \textbf{ in return}(M_A); Q_{1A},
\textbf{event } e_B(\text{sid}(\widetilde{msg}_B), k_B); \textbf{return}(M_B); Q_{1B}]
\]

\[
S_2 = O_2() := k \overset{R}{\leftarrow} T; \textbf{return}(); (Q_{2A} \mid Q_{2B})
\]

\[
S_{2!} = \text{AddReplSid}(\widetilde{i} \leq \widetilde{n}, O'_2, T_{sid}, S_2)
\]

where

1. \(O'_2, k'_A, e_A, e_B\) do not occur elsewhere in \(S_1, S_{2!}\);
2. \(S_1\) and \(S_{2!}\) have no common variable, oracle, event, table;
3. \(S_1\) and \(S_{2!}\) do not contain \textbf{newOracle} nor oracle calls;
4. and there is no \textbf{defined} condition in \(S_2\).
Main composition theorem

\[ S_1 = C[\text{event } e_A(\text{sid}(\tilde{msg}_A), k_A, i); \text{let } k'_A = k_A \text{ in return}(M_A); Q_{1A}, \]
\[ \text{event } e_B(\text{sid}(\tilde{msg}_B), k_B); \text{return}(M_B); Q_{1B}] \]
\[ S_2 = O_2() := k \xleftarrow{\text{R}} T; \text{return}(); (Q_{2A} \mid Q_{2B}) \]
\[ S_2! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, O'_2, T_{\text{sid}}, S_2) \]

where

1. \( O'_2, k'_A, e_A, e_B \) do not occur elsewhere in \( S_1, S_2! \);
2. \( S_1 \) and \( S_2! \) have no common variable, oracle, event, table;
3. \( S_1 \) and \( S_2! \) do not contain \texttt{newOracle} nor oracle calls;
4. and there is no \texttt{defined} condition in \( S_2 \).

\textit{sid is a function that takes a sequence of messages and returns a session identifier of type } T_{\text{sid}}
Theorem \((S_1, S_2)\)

\[
S_1 = C[\text{event } e_A(\text{sid}(\tilde{\text{msg}}_A), k_A, \tilde{i}); \text{let } k'_A = k_A \text{ in return}(M_A); Q_{1A}, \text{event } e_B(\text{sid}(\tilde{\text{msg}}_B), k_B); \text{return}(M_B); Q_{1B}]
\]

\[
S_2 = O_2() := k \xleftarrow{R} T; \text{return}(); (Q_{2A} \mid Q_{2B})
\]

\[
S_2! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, O'_2, T_{\text{sid}}, S_2)
\]

where
\begin{enumerate}
\item \(O'_2, k'_A, e_A, e_B\) do not occur elsewhere in \(S_1, S_2!\);  
\item \(S_1 \text{ and } S_2!\) have no common variable, oracle, event, table;  
\item \(S_1 \text{ and } S_2!\) do not contain \texttt{newOracle} nor oracle calls;  
\item and there is no defined condition in \(S_2\).
\end{enumerate}
Main composition theorem

Theorem \((S_1 \equiv S_2)\)

\[
S_1 = C[\text{event } e_A(\text{sid}(\bar{m}_A), k_A, i); \text{let } k'_A = k_A \text{ in return}(M_A); Q_{1A},
\text{event } e_B(\text{sid}(\bar{m}_B), k_B); \text{return}(M_B); Q_{1B}]
\]

\[
S_2 = O_2() := k \xleftarrow{R} T; \text{return}(); (Q_{2A} \mid Q_{2B})
\]

\[
S_2! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, O'_2, T_{\text{sid}}, S_2)
\]

where

1. \(O'_2, k'_A, e_A, e_B\) do not occur elsewhere in \(S_1, S_2!\);
2. \(S_1\) and \(S_2!\) have no common variable, oracle, event, table;
3. \(S_1\) and \(S_2!\) do not contain \texttt{newOracle} nor oracle calls;
4. and there is no \texttt{defined} condition in \(S_2\).

\(\bar{m}_B\) is a sequence of variables received or returned by \(C\) above the second hole.
Main composition theorem

**Theorem (\(S_1\) and \(S_2!\))**

\[
S_1 = C[\text{event } e_A(\text{sid}(\tilde{\text{msg}}_A), k_A, \tilde{i}); \text{let } k'_A = k_A \text{ in return}(M_A); Q_{1A}, \text{event } e_B(\text{sid}(\tilde{\text{msg}}_B), k_B); \text{return}(M_B); Q_{1B}]
\]

\[
S_2 = O_2() := k \xleftarrow{\text{R}} T; \text{return}(); (Q_{2A} \mid Q_{2B})
\]

\[
S_2! = \text{AddReplSid}(\tilde{i} \leq \tilde{n}, O'_2, T_{\text{sid}}, S_2)
\]

where

1. \(O'_2, k'_A, e_A, e_B\) do not occur elsewhere in \(S_1, S_2!\);
2. \(S_1\) and \(S_2!\) have no common variable, oracle, event, table;
3. \(S_1\) and \(S_2!\) do not contain \texttt{newOracle} nor oracle calls;
4. and there is no defined condition in \(S_2\).
**Main composition theorem**

**Theorem** \( (S_{\text{composed}}) \)

Let \( Q'_{2A} = \text{AddIdxSid}(\tilde{i} \leq \tilde{n}, x : T_{\text{sid}}, Q_{2A}) \)
and \( Q'_{2B} = \text{AddIdxSid}(\tilde{i}' \leq \tilde{n}', x : T_{\text{sid}}, Q_{2B}) \).

Let 

\[
S_{\text{composed}} = C[\text{event } e_A(\text{sid}(\sim\text{msg }_A), k_A, \tilde{i}); \text{return}(M_A); \\
(Q_{1A} \mid Q'_{2A}\{k_A/k, \text{sid}(\sim\text{msg }_A)/x\})], \\
\text{event } e_B(\text{sid}(\sim\text{msg }_B), k_B); \text{return}(M_B); \\
(Q_{1B} \mid Q'_{2B}\{k_B/k, \text{sid}(\sim\text{msg }_B)/x\})]
\]
Main composition theorem

Theorem (First conclusion)

If $S_1$ satisfies

- secrecy of $k'_A$ with public variables $V \ (V \subseteq \text{var}(S_1) \setminus \{k_A, k'_A\})$,
- injective authentication of $A$ to $B$:
  \[
  \text{inj-event}(e_B(sid, k)) \implies \text{inj-event}(e_A(sid, k, \tilde{i}))
  \]
  with public variables $V \cup \{k'_A\}$,
- single $e_A$ for each session identifier:
  \[
  \text{event}(e_A(sid, k_1, \tilde{i}_1)) \land \text{event}(e_A(sid, k_2, \tilde{i}_2)) \implies \tilde{i}_1 = \tilde{i}_2
  \]
  with public variables $V \cup \{k'_A\}$,

then we can transfer security properties from $S_{2!}$ to $S_{\text{composed}}$.

Up to renumbering of variable indices,

$S_{\text{composed}}$ with the events of $S_1$ removed

is indistinguishable with public variables $V \cup (\text{var}(S_2) \setminus \{k\})$

from an evaluation context interacting with $S_{2!}$. 
Main composition theorem

Theorem (Second conclusion)

We can transfer security properties from $S_1$ to $S_{\text{composed}}$, provided they are proved with public variables $k'_A, k_B$.

$S_{\text{composed}}$ is indistinguishable with public variables $\text{var}(S_{\text{composed}}) \setminus \{k'_A\}$ from an evaluation context interacting with $S_1$ with access to $k'_A, k_B$. 
Further results in the paper

- **Exact security.**
- **New:** Shared hash oracles between the key exchange and the protocol that uses the key.
- **New:** Variant with non-injective authentication.
- **New:** Variant for modified ClientHello messages.

The paper was written using a syntax with channels instead of oracles, hence the theorems had to be adapted accordingly.
Transport Layer Security (TLS) 1.3
joint work with Karthikeyan Bhargavan and Nadim Kobeissi

- Next version of the most popular secure channel protocol.
  - Completely redesigned from TLS 1.2
  - Standardized after 28 drafts
Transport Layer Security (TLS) 1.3
joint work with Karthikeyan Bhargavan and Nadim Kobeissi

- Next version of the most popular secure channel protocol.
  - Completely redesigned from TLS 1.2
  - Standardized after 28 drafts

- Why did we need a new protocol?
  - Security: remove broken legacy crypto constructions
<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
  - Standardized after 28 drafts

- **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
Transport Layer Security (TLS) 1.3

- Next version of the most popular secure channel protocol.
  - Completely redesigned from TLS 1.2
  - Standardized after 28 drafts

- 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.
Summary of TLS 1.3

**Client**

<table>
<thead>
<tr>
<th>ClientHello</th>
<th>ServerHello</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ key_share*</td>
<td>+ key_share*</td>
</tr>
<tr>
<td>+ pre_shared_key*</td>
<td>{EncryptedExtensions}</td>
</tr>
<tr>
<td>(Application*)</td>
<td>{CertificateRequest*}</td>
</tr>
<tr>
<td></td>
<td>{Certificate*}</td>
</tr>
<tr>
<td></td>
<td>{CertificateVerify*}</td>
</tr>
<tr>
<td></td>
<td>{Finished}</td>
</tr>
<tr>
<td></td>
<td>[ApplicationData*]</td>
</tr>
</tbody>
</table>

**Server**

<table>
<thead>
<tr>
<th>HelloRetryRequest</th>
<th>+ important extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ key_share</td>
<td></td>
</tr>
</tbody>
</table>

* may be absent

Parentheses = encryption:
- () under key derived from client early traffic secret
- {} under key derived from handshake traffic secret
- [] under key derived from application traffic secret
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.

- 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

cats  sats  ems  resumption_secret

cats  sats  ems  cets

updated ts
Key schedule (Draft-18, excerpt)

PSK $\rightarrow$ HKDF-Extract

Early Secrets

0

Derive-Secret(., “external psk binder key” | “resumption psk binder key”, “”)

$= binder_key$

Derive-Secret(., “client early traffic secret”, ClientHello)

$= client\_early\_traffic\_secret (cets)$

(EC)DHE $\rightarrow$ HKDF-Extract

Handshake Secret
Assumptions (1)

- **Diffie-Hellman:**
  - gap Diffie-Hellman (GDH)
    - needed in particular for 0.5-RTT
  - 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 $\text{len}_H() || "\text{TLS 1.3, } \| / \| h || 0x01$.
    - 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 $x \mapsto \text{HMAC-}H^{\text{kdf}_0}(x)$ are independent random oracles.
  - HMAC-H is a PRF, for keys different from $0^{\text{len}H}$ and $\text{kdf}_0$.
- **Authenticated Encryption:** IND-CPA and INT-CTXT provided the same nonce is never used twice with the same key.
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)

- Mutual injective 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 $cats$, $sats$, 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.

  The previous properties are injective.

- Key secrecy: the keys
  - $cats$, $ems$, $psk'$ client side, when the client terminates a honest session;
  - $sats$ 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)

- **Unique accept event for each session identifier.**
  - The server never accepts twice with the honest client and the same messages until the server Finished message.
  - The client never accepts twice with the honest server and the same messages until the client Finished message.

- **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 $x \mapsto \text{HMAC-H}^{kd_f_0}(x)$.
- 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 $\text{enc}^{k_t}(\text{psk})$; 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

- Additionally, we prove forward secrecy wrt. to the compromise of PSK for PSK-DHE (requires CryptoVerif 2.02).

- **Weaker properties for 0-RTT:**
  - **Key authentication:** No authentication for cets:
    - several binders, and only one of them is checked;
    - the adversary can alter the others, yielding a different cets server-side.
  - **Replay prevention:** No replay protection for cets.
  - **Secrecy of keys:** The keys cets 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 $cets$ are *secret*: indistinguishable from independent random values.
- **Server-side**:
  - If the received ClientHello message has been sent by the client, then have *non-injective authentication of client to server*: this session matches a session of the client with same key $cets$.
  - Otherwise,
    - If the ClientHello message has been received before, then the key $cets$ computed by the server is the same as in the previous session with the same ClientHello message.
    - Otherwise, the key $cets$ computed by the server is indistinguishable from a fresh random value, independent from other keys.
Security of the record protocol

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

- **Key secrecy**: The updated traffic secret is secret.
- **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.
- **Injective message authentication**: Every time a message $m$ is decrypted by the receiver with a counter $c$, the message $m$ has been encrypted and sent by an honest sender with the same counter $c$. 
Composition

Handshake without pre-shared key

Handshake with pre-shared key

Record protocol

\textit{cats, sats, ems, resumption\_secret, cets, updated ts}
Conclusion

- Composition theorems for **CryptoVerif**
  - computational
  - easy to apply when the protocol pieces are proved secure in **CryptoVerif**
  - flexible: hash oracles, injective and non-injective authentication

- Application to **TLS 1.3**
  - important protocol
  - would be out of scope of **CryptoVerif** without composition because of loops

- Applicable to other protocols
Future directions

- Composition theorems could be proved for other tools, such as EasyCrypt.
- We could automate the verification of the assumptions of our theorems and the computation of the composed protocol.
  - Automating the TLS case study would be more difficult (recursive composition).
- We could consider composition with a key exchange protocol that already uses the key.