A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello

Karthikeyan Bhargavan\textsuperscript{1}
Vincent Cheval\textsuperscript{1}
Christopher Wood\textsuperscript{2}

\textsuperscript{(1)} INRIA Paris
Equipe Prosecco

\textsuperscript{(2)} Cloudflare
TLS Deployment Scenario
The basic TLS 1.3 handshake

ClientHello
+Key Share
+Signature Algorithm
+Pre share key

ServerHello
+Key Share
+Pre share key

{EncryptedExtensions}
{CertificateRequest}
{Certificate}
{CertificateVerify}
{Finished}
[Application Data]

{Certificate}
{CertificateVerify}
{Finished}
[Application Data]

[Application Data]

In green: Not always sent
{ X } : Encrypted with Handshake traffic key
[ X ] : Encrypted with Application traffic key

Diffie-Hellman key exchange
Several features

- Negotiating Connection Parameters: HelloRetryRequest
- Certificate-based Client Authentication
- Pre-Shared Keys and Tickets
- 0RTT
- Post Handshake Authentication
- Other TLS extensions (e.g. SNI)

Verifying TLS requires to consider many scenarios
## Security goals

<table>
<thead>
<tr>
<th>Authentication and Integrity Goals</th>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Authentication</td>
<td>Key Secrecy</td>
</tr>
<tr>
<td>Client Authentication</td>
<td>Key Indistinguishability</td>
</tr>
<tr>
<td>Key and Transcript Agreement</td>
<td>1RTT Data Forward Secrecy</td>
</tr>
<tr>
<td>Data Stream Integrity</td>
<td>0RTT Data Secrecy</td>
</tr>
<tr>
<td>Key Uniqueness</td>
<td></td>
</tr>
<tr>
<td>Downgrade Resilience</td>
<td></td>
</tr>
</tbody>
</table>
## Security goals

<table>
<thead>
<tr>
<th>Authentication and Integrity Goals</th>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Authentication (1, 3, 4)</td>
<td>Key Secrecy (1, 2, 3, 4)</td>
</tr>
<tr>
<td>Client Authentication (1, 3, 4)</td>
<td>Key Indistinguishability (1)</td>
</tr>
<tr>
<td>Key and Transcript Agreement (1, 3, 4)</td>
<td>1RTT Data Forward Secrecy (1, 3, 4)</td>
</tr>
<tr>
<td>Data Stream Integrity (1, 2, 3, 4)</td>
<td>0RTT Data Secrecy (1, 2, 3, 4)</td>
</tr>
<tr>
<td>Key Uniqueness (3, 4)</td>
<td></td>
</tr>
<tr>
<td>Downgrade Resilience (4)</td>
<td></td>
</tr>
</tbody>
</table>

Automated verification to the rescue

1. CryptoVerif
2. F*
3. Tamarin
4. ProVerif
Security goals

Authentication and Integrity Goals

- Server Authentication (1,3,4)
- Client Authentication (1,3,4)
- Key and Transcript Agreement (1,3,4)
- Data Stream Integrity (1,2,3,4)
- Key Uniqueness (3,4)
- Downgrade Resilience (4)

Confidentiality

- Key Secrecy (1,2,3,4)
- Key Indistinguishability (1)
- 1RTT Data Forward Secrecy (1,3,4)
- 0RTT Data Secrecy (1,2,3,4)

Automated verification to the rescue

1. CryptoVerif
2. F*
3. Tamarin
4. ProVerif

These models do not cover all features
Security goals

Privacy

Client Identity Privacy
Client Unlinkability
Server Extension Privacy
Client Extension Privacy
Server Identity Privacy
Security goals

Privacy

Client Identity Privacy
Client Unlinkability
Server Extension Privacy
Client Extension Privacy
Server Identity Privacy

No automated proofs
Extension in ClientHello
SNI in ClientHello
Security goals

Privacy

- Client Identity Privacy
- Client Unlinkability
- Server Extension Privacy
- Client Extension Privacy
- Server Identity Privacy

Encrypted Client Hello guarantees all these privacy goals

No automated proofs

Extension in ClientHello

SNI in ClientHello
Goal: Privacy of the identity of the backend server

Main idea: Encrypt sensitive informations (e.g. server identity of the backend server) with a public key of the client-facing server
Not so easy: Several previous designs were vulnerable

First draft: Encrypt the SNI and ClientHello.random

- Client (C)
  - CH(...[G, g^e]...)
  - Computes:
    - $h_s = kdf_{hs}(kdf_{0, g^{e^y}})$
    - $ms, k_{h,c}, k_{h,s}, \cdots = kdf_{ms}(hs, tx_1)$
  - $\text{Enc}_{k_{h,s}}(\text{Extensions}(\ldots), \text{Certificate}(S,:pk_S))$
  - Learns that $C$ tried to connect to $S$ from $\text{Certificate}(S',:pk'_S)$

- Adversary (A)
  - CH($cr, F, [G, g^e]$)
  - Computes:
    - $h_s = kdf_{hs}(kdf_{0, g^{e^y}})$
    - $ms, k_{h,c}, k_{h,s}, \cdots = kdf_{ms}(hs, tx_1)$

- Server (F, S, S')
  - Long-term Keys: $(dk_F, ek_F), (sk_S, pk_S), (sk'_S, pk'_S)$
  - $\text{Hpke}^{ek_F}(S)[cr]$
  - $\text{ServerHello}(sr, G, g^y)$

Hybrid Public Key Encryption
Not so easy: Several previous designs were vulnerable

First draft: Encrypt the SNI and ClientHello.random

Main idea: Encrypt the whole Client Hello destined for the backend server (inner) and bind it with the Client Hello for the Client-Facing server (outer)
Not so easy: HelloRetryRequest

Client (C)  Adversary (A)  Server (F, S, S')

Inner Client Hello

Long-term Keys: (dkF, ekF), (skS, pkS), (skS', pkS')

New Inner Client Hello independent from the first one

$\text{CH}(cr, F, [(G_0, g^x), G_1], \text{hpke}^{ek_F}(\text{CH}(cr_1, S, [(G_0, g^x), G_1])))$

$\text{HRR}(G_1, \text{accept}^{en_c}_{\text{serv}})$

$\text{CH}(cr, F, [(G_1, g^x')], \text{hpke}^{ek_F}(\text{CH}(cr_1', S', [(G_1, g^x')])))$

$\text{SH}(sr, G, g^y)$

Computes:

$hs = \text{kdf}_{hv}(es, g^{x'})$

$ms, k_{h,s}, \ldots = \text{kdf}_{ms}(hs, tx_3)$

$\text{tx}_4 \overset{\text{enc}}{\rightarrow} \text{enck}_{hs}(\text{Extensions}(...), \text{Certificate}(S, pk_S))$

$\text{tx}_5 \overset{\text{enc}}{\rightarrow} \text{enck}_{hs}(\text{CertVerify}(\text{sign}_{sk_S}(H(\text{tx}_4))))$

Learns that C tried to connect to S
Not so easy: HelloRetryRequest

The encryption of the second Inner Client Hello must be linked to the first Inner Client Hello.
Encrypted Client Hello (ECH)

The context $ctx$ is updated after each encryption ($ctx', ctx''$)

Client ($C$)

Long-term Keys: $(sk_C, pk_C, psk_{C,S})$

Supports protocol parameters: $[[TLS1.3+ECH, TLS1.3], DHE[G_0, G_1], H(), enc()]$

Generates $(x, g^x), (x, g^x)$ in $G_0$ and computes:

$C, ctx = hpkeSetupS(ek_F)$
$ech, ctx' = hpkeSeal(ctx, ClientHello(cr_1, S, [(G_0, g^{x'}), G_1]))$

$tz_0$ $\rightarrow$ ClientHello($cr, F, [(G_0, g^x), G_1], (C, ech)$)

$tz_0$ $\rightarrow$ HelloRetryRequest($G_1, accept^{CTX}(tz_1)$)

$tx_1$ $\rightarrow$ ClientHello($cr', F, [(G_1, g^{x'}), G_1], (C, ech)$)

Generates $(x', g^{x'})$, $(x', g^{x'})$ in $G_1$
Computes: $es = kdf_0$ and encrypts
$ech', ctx'' = hpkeSeal(ctx', ClientHello(cr_1', S, [(G_1, g^{x''})])$

$tz_2$ $\rightarrow$ ClientHello($cr, F, [(G_1, g^{x'}), ech']$

$tz_3$ $\rightarrow$ ServerHello($sr, G_1, g^{x'}, accept^{CTX}(tz_3)$)

Computes:
$hs = kdf_{hs}(es, g^{x'})$
$ms, k_{h,c}, k_{h,a}, k_{m,c}, k_{m,a} = kdf_{ms}(hs, tz_3)$

Server ($S$, $S'$, $F$)

Long-term Keys: $(sk_S, pk_S, (sk'_S, pk'_S), (ek_F, ek_P))$

Supports protocol parameters: $[[TLS1.3+ECH, TLS1.3], DHE[G_1], H(), enc()]$

Computes: $ctx = hpkeSetupR(C, sk_F)$ and decrypts
ClientHello($cr, S, [(G_0, g^{x'}), G_1]), ctx' = hpkeOpen(ctx, ech)$

$tx_0$ $\rightarrow$ ServerHello($sr, G_1, g', accept^{CTX}(tz_1)$)

$tz_1$ $\rightarrow$ ClientHello($cr_1, S, [(G_0, g^{x'}), G_1])$

$tx_2$ $\rightarrow$ Decrypts ClientHello($cr', S, [(G_1, g^{x''})]), ctx'' = hpkeOpen(ctx', ech')$
Generates: $(y, g^y)$ in $G_1$, and computes: $es = kdf_0$

$tz_3$ $\rightarrow$ Computes:
$hs = kdf_{hs}(es, g^{y'})$
$ms, k_{h,c}, k_{h,a}, k_{m,c}, k_{m,a} = kdf_{ms}(hs, tz_3)$
Attacker model

The attacker can...

- Read / Write
- Intercept

But they do not...

- Break cryptography
- Use side channels

Dolev-Yao models
Concurrent systems where dishonest parties have complete control over network communication

*but* cryptography is idealised

Automated Verification Tool: ProVerif
Our model

Focus only on TLS 1.3 (no version negotiation)
Model all features presented before (e.g. HRR, PHA, PSK, Ticket, ECH, 1RTT and 0RTT Data)
Model all security properties presented before (i.e. Authentication, Integrity, Confidentiality and Privacy goals)

Proving all properties with all features is too taxing on ProVerif in computation time or memory consumption
OOT = 48H and OOM = 100GB
Our model

Focus only on TLS 1.3 (no version negotiation)
Model all features presented before (e.g. HRR, PHA, PSK, Ticket, ECH, 1RTT and 0RTT Data)
Model all security properties presented before (i.e. Authentication, Integrity, Confidentiality and Privacy goals)

Proving all properties with all features is too taxing on ProVerif in computation time or memory consumption
OOT = 48H  and  OOM = 100GB

Parametrized model

Simple configuration file allows us to activate/deactivate:

- Features
- Compromised keys
- Server and client behavior

621 runs of ProVerif
Our results (Authentication, Integrity, Confidentiality)

✓: Feature enabled  ×: Feature disabled

<table>
<thead>
<tr>
<th>Property</th>
<th>1-RTT</th>
<th>HRR</th>
<th>CC</th>
<th>PHA</th>
<th>PSK-DHE</th>
<th>TKT</th>
<th>0-RTT</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>10h7m</td>
</tr>
<tr>
<td>SEC, UNIQ</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>2h48m</td>
</tr>
<tr>
<td>SEC0</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>55m</td>
</tr>
<tr>
<td>FS, INT</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>3h40m</td>
</tr>
<tr>
<td>CAUTH</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>2h39m</td>
</tr>
<tr>
<td>SAUTH, AGR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>3h26m</td>
</tr>
<tr>
<td>DOWN</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>34h16m</td>
</tr>
</tbody>
</table>

Sanity Checks

Security preservation

Downgrade resilience w.r.t. ECH

Computation time
Assumptions for Privacy of Server Identity

1. HPKE private key of Client-facing server $f_{s^*}$ is uncompromised
   - If not: The can directly decrypt the ECH extension to obtain the identity of the backend server

2. $BS_1$ and $BS_2$ both have a certificate long term key or none of them have one.
   - If not: The basic handshake where the server must send its certificate will only succeed in one of the scenarios

3. $A$ share a (different, uncompromised) PSK with both $BS_1$ and $BS_2$ or with neither of them.
   - If not: The number of messages sent will differ
### Our results (Privacy)

For Privacy properties, 1RTT and 0RTT are disabled

- ✓: Feature enabled
- ✗: Feature disabled

<table>
<thead>
<tr>
<th>Property</th>
<th>HRR</th>
<th>CC</th>
<th>PHA</th>
<th>PSK-DHE</th>
<th>TKT</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IND, CIP UNL, S-EXT</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>17H15</td>
</tr>
<tr>
<td>CIP, UNL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>10h10m</td>
</tr>
<tr>
<td>IND</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>21h16m</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>12h47</td>
</tr>
<tr>
<td>ECH</td>
<td>SIP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>24h27m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1h13m</td>
</tr>
<tr>
<td>CIP, UNL</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>21h42m</td>
</tr>
<tr>
<td>S-EXT, C-EXT</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>21h20m</td>
</tr>
</tbody>
</table>

Privacy properties requires more time and memory

Ongoing work: Improve ProVerif to reduce memory consumption
Thank you!

Questions?