



Aalto University

Network Security: TLS 1.3 handshake

Tuomas Aura, Aalto University

CS-E4300 Network security

Outline

- TLS 1.3 full handshake: 1-RTT
- Security properties, identity protection

Please refer to the
Information Security
course for an
introduction to TLS

TLS 1.3 full handshake

Client

ClientHello

+ key_share*

+ signature_algorithms*

+ supported_groups*

+ server_name*

+ certificate_authorities*

4. Client authentication
(typically omitted)

{Certificate*}
{CertificateVerify*}
{Finished}
[Application data]

Server

ServerHello

+ key_share*

{EncryptedExtensions}
{CertificateRequest*}
{Certificate*}
{CertificateVerify*}
{Finished}
[ApplicationData*]

[Application data]

1. Parameter negotiation

2. DHE or ECDHE key exchange

3. Server authentication

5. Key confirmation

6. Protected session data

----->

<-----

----->

<----->

TLS 1.3 full handshake

1. C → S: N_C , supported_versions, supported_groups, signature_algorithms, cipher_suites, server_name, certificate_authorities, g^x
 2. S → C: N_S , version, cipher_suite, g^y
EncryptedExtensions
 $Cert_S, Sign_S(TH)$
 $HMAC_{K_{fks}}(TH)$
 3. C → S: $Cert_C, Sign_C(TH)$
 $HMAC_{K_{fkc}}(TH)$
- encrypted with K_{shts}
- encrypted with K_{chts}

N_C, N_S = client and server random

$Cert_C, Cert_S$ = certificate chain

TH = transcript hash i.e. hash of all previous messages

Exchange keys $K_{chts}, K_{shts}, K_{fkc}, K_{fks}$ session keys K_{cats}, K_{sats} derived from g^{xy} and TH

TLS 1.3 algorithms

- Small number of modern cipher suites
- AEAD ciphers: encryption and authentication always together
- Perfect forward secrecy required
 - Only ephemeral key exchanges: DHE or ECDHE
 - Old RSA handshake is not supported

1-RTT handshake

Client

ClientHello

- + key_share*
- + signature_algorithms*
- + supported_groups*
- + server_name*
- + certificate_authorities*

Client does not know which groups the server supports but makes a guess

Server

ServerHello

- + key_share*
- {EncryptedExtensions}
- {CertificateRequest*}
- {Certificate*}
- {CertificateVerify*}
- {Finished}
- [ApplicationData*]

- {Certificate*}
- {CertificateVerify*}
- {Finished}
- [Application data]

[Application data]

1-RTT handshake

- TLS 1.3 handshake causes only one round-trip delay
 - Client can send HTTP request (application data) right after client Finished
 - TLS 1.2 and most other key-exchange protocols require two RTT
 - Important for page load times in web browsing
- However, TCP + TLS 1.3 together cause 2-RTT latency
 - QUIC avoids this because it runs over UDP
- Sometimes TLS 1.3 handshake takes two RTT:
 - If server does not support the group of key_share in ClientHello, server sends HelloRetryRequest to ask for a different curve
 - DTLS server under DoS attack can send a Cookie in HelloRetryRequest

Key derivation

Inputs to key derivation:

1. PSK (external PSK or resumption PSK)
 2. DHE/ECDHE secret
 3. Transcript of handshake messages, up to the point where the key is derived
- } one or both, as available

Keys:

- client_early_traffic_secret → used to derive AEAD keys for early data in 0-RTT (...)
- client/server_handshake_traffic_secret → used to derive AEAD keys for handshake messages {...} and Finished HMAC keys
- client/server_application_traffic_secret_N → used to derive AEAD encryption keys for post-handshake application data and messages [...]
- resumption_master_secret and ticket_nonce → derive resumption PSK
- exporter_master_secret → used to create keys for the application layer

Post-handshake client authentication

- Server can request client authentication any time, either **during** or **after** the TLS handshake
- **Post-handshake** client authentication **allows time for user action, such as inserting a smartcard**
 - Application can give user more access rights after the authentication

References

- TLS 1.3, [RFC 8446](#)
- The New Illustrated TLS Connection, <https://tls13.ulfheim.net/>

Exercises

- Use a network sniffer (e.g. Netmon, Ethereal) to look at TLS handshakes. Can you spot a full handshake and session resumption? Can you see the plaintext SNI?
- Compare TLS 1.3 and TLS 1.2 handshakes in network trace: Can you see the difference in round-trips, identity protection?
- How would you modify the TLS 1.3 handshake to improve identity protection? Learn about PEAP. How does PEAP protect the client identity?
- Consider removing message fields from the handshake. How does each message field contribute to security?
- Why have the supported and mandatory-to-implement cipher suites in TLS changed over time?
- Why did most web servers for a long time prefer the RSA handshake?
- One reason why the RSA handshake is no longer supported in TLS 1.3 is that it does not provide PFS. Is it possible to implement PFS without Diffie-Hellman?
- Find applications that could benefit significantly from the 0-RTT handshake. Is there any cost to deploying it?
- What problems arise if you want to set up multiple secure (HTTPS) web sites behind a NAT or on virtual servers that share one IP address? How do TLS 1.3 and TLS 1.2 solve this issue?
- If an online service (e.g. webmail) uses TLS with server-only authentication to protect passwords, is the system vulnerable to offline password cracking?