Overview of TLS v1.3
What’s new, what’s removed and what’s changed?
• Andy Brodie
  – Worldpay Principal Design Engineer.
  – Based in Cambridge, UK.
  – andy.brodie@owasp.org

• Neither a cryptographer nor a mathematician!
  – This means no maths in this presentation.
• History & Background.
• What’s Been Removed.
• What’s New & Changed.
  – Cipher Suites.
  – Handshake Changes.
  – Hashed-Key Derivation Function.
  – Session Resumption.
• Summary.
The Goals and Basics of TLS

HISTORY & BACKGROUND
<table>
<thead>
<tr>
<th>When</th>
<th>Who</th>
<th>What</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>Netscape</td>
<td>SSL 1.0 designed.</td>
<td>Never published as security flaws were found internally.</td>
</tr>
<tr>
<td>1995</td>
<td>Netscape</td>
<td>SSL v2.0 published.</td>
<td>Flaws found pretty quickly, which led to...</td>
</tr>
<tr>
<td>1996</td>
<td>Netscape</td>
<td>SSL v3.0 published.</td>
<td>SSL becomes ubiquitous.</td>
</tr>
<tr>
<td>1999</td>
<td>IETF</td>
<td>TLS v1.0 published (SSL v3.1)</td>
<td>Incremental fixes, political name change and IETF ownership.</td>
</tr>
<tr>
<td>2006</td>
<td>IETF</td>
<td>TLS v1.1 published (SSL v3.2)</td>
<td>Incremental fixes and capabilities.</td>
</tr>
<tr>
<td>2008</td>
<td>IETF</td>
<td>TLS v1.2 published (SSL v3.3)</td>
<td>What we should all be using!</td>
</tr>
<tr>
<td>2014</td>
<td>IETF</td>
<td>TLS v1.3 draft 1 (SSL v3.4)</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>IETF</td>
<td>TLS v1.3 draft 23</td>
<td>Expires July 15</td>
</tr>
</tbody>
</table>
A Client and Server can have a secure conversation over an insecure medium having never met before.
What is a secure conversation?

- **Privacy**
  - Conversation must be encrypted.
  - Prevent eavesdropping attacks.

- **Integrity**
  - Client & Server must be able to detect message tampering.
  - Prevent Man In The Middle (MITM) attacks.

- **Authentication**
  - Client needs to trust they’re talking to the intended server.
  - Prevent impersonation attacks.
• **Privacy**
  – Symmetric key encryption for application data.
  – Typically Advanced Encryption Standard (AES).

• **Integrity**
  – Authenticated Encryption with Additional Data (AEAD).
  – Usually AES-GCM (Galois/Counter Mode) cipher mode.

• **Authentication**
  – X509 certificates signed by a mutually trusted third party.
  – Typically server authenticated only.
Flow of messages in a TLS conversation

- Open Socket
  - Handshake
    - Application Data
      - Alert
        - Close Socket
Flow of messages in a TLS conversation

- **Handshake**
  - Agree a cipher suite.
  - Agree a master secret.
  - Authentication using certificate(s).

- **Application Data**
  - Symmetric key encryption.
  - AEAD cipher modes.
  - Typically HTTP.

- **Alerts**
  - Graceful closure, or
  - Problem detected.

TLS V1.3
Key Goals of TLS v1.3:

- **Clean up** - Remove unsafe or unused features.
- **Security** - Improve security w/modern techniques.
- **Privacy** - Encrypt more of the protocol.
- **Performance** – 1-RTT and 0-RTT handshakes.
- **Continuity** – Backwards compatibility.
WHAT’S REMOVED IN TLS V1.3?
What’s removed in TLS v1.3

- **Key Exchange**
  - RSA

- **Encryption algorithms:**
  - RC4, 3DES, Camellia.

- **Cryptographic Hash algorithms:**
  - MD5, SHA-1.

- **Cipher Modes:**
  - AES-CBC.

- **Other features:**
  - TLS Compression & Session Renegotiation.
  - DSA Signatures (ECDSA ≥ 224 bit).
  - ChangeCipherSpec message type & “Export” strength ciphers.
  - Arbitrary/Custom (EC)DHE groups and curves.
This has mitigated quite a few attacks...

RC4
- Roos’s Bias 1995
- Fluhrer, Martin & Shamir 2001
- Klein 2005
- Combinatorial Problem 2001
- Royal Holloway 2013
- Bar-mitzvah 2015
- NOMORE 2015

RSA-PKCS#1 v1.5 Encryption
- Bleichenbacher 1998
- Jager 2015
- DROWN 2016

Renegotiation
- Marsh Ray Attack 2009
- Renegotiation DoS 2011
- Triple Handshake 2014

3DES
- Sweet32

AES-CBC
- Vaudenay 2002
- Boneh/Brumley 2003
- BEAST 2011
- Lucky13 2013
- POODLE 2014
- Lucky Microseconds 2015

Compression
- CRIME 2012

MD5 & SHA1
- SLOTH 2016
- SHAAttered 2017
WHAT’S NEW AND CHANGED?
What’s New and Changed?

• Cipher Suites.
• Handshake.
• Hashed-Key Derivation Function (HKDF).
• Key Schedule.
• Sessions.
CIPHER SUITES
TLS v1.2 provides 37 Cipher Suites

- TLS 1.2 specifies 37 cipher suites.
  - Add previous versions in: 319 cipher suites.
• TLS v1.3 supports 5 cipher suites.
  – TLS_AES_128_GCM_SHA256
  – TLS_AES_256_GCM_SHA384
  – TLS_CHACHA20_POLY1305_SHA256
  – TLS_AES_128_CCM_SHA256
  – TLS_AES_128_CCM_8_SHA256
What happens to key exchange and authentication then?

• Key Exchange algorithms:
  – **DHE & ECDHE**
    • Only 5 ECDHE curve groups supported
    • Only 5 DHE finite field groups supported
  – Pre-Shared Key (PSK)
  – PSK with (EC)DHE

• Digital Signature (Authentication) algorithms:
  – RSA (PKCS#1 variants)
  – ECDSA / EdDSA
HANDSHAKE CHANGES
The handshake has three goals:
- Agree a cipher suite.
- Agree a master secret.
- Establish trust between Client & Server.

Optimise for the most common use cases.
- Everyone* wants a secure conversation.
- Same cipher suites used across websites repeatedly.
- Clients connect to the same sites repeatedly.

* ok, *almost* everyone!
TLS 1.2 Handshake

Client

- Client Hello
- server hello
- certificate
- server key exchange
- certificate request
- hello done

Server

- certificate
- key exchange
- certificate verify
- change cipher spec
- finished

- change cipher spec
- finished
Three Stages of a TLS 1.3 Handshake

Key Exchange

Server Parameters

Authentication
Client now makes assumptions about server support.

- **Client sends:**
  - Cipher Suite options.
  - List of supported groups/curves.
  - (EC)DHE Key Share(s).

- **Server sends:**
  - Cipher suite selection.
  - (EC)DHE Key Share

- **Client and Server now share a key.**
• Server sends:
  – Encrypted Extensions
    • Server Name
    • Message Length
    • ...and optionally many more
  – Certificate Request
    • Supported signature algorithms.
Client now makes assumptions about server support.

- Server sends:
  - Certificate.
  - Proof of private key possession.
  - Finished.
  - Application Data

- Client responds:
  - Certificate.
  - Proof of private key possession.
  - Finished.
Efficiency Gains
GENERATING KEYS USING HKDF
• TLS <= v1.2 defines PRF algorithm.

• TLS v1.3 replaces this with HKDF.
  – HKDF encapsulates how TLS uses HMAC.
  – Re-used in other protocols.
  – Separate cryptographic analysis already done.

• Provides 2 functions:
  – Extract - create a pseudo-random key from inputs.
  – Expand - create more keys from the extract output.

• HMAC is integral to HKDF.
  – HMAC requires the Cryptographic Hash algorithm specified in the cipher suite (SHA256 or SHA384).
How the PRF is implemented

\[ \text{PRF(secret, label, seed)} \]

\[ \text{P_HASH(secret, label + seed)} \]

\[ \text{HMAC(SHA-256)} \]

\[ \text{label + seed} \]

\[ \text{Key Material} \]
TLS <= v1.2 Creating Key Material from a master secret

Pre-master Secret

Master Secret

>= 46 bytes

48 bytes

Key Material

Client MAC Key

Server MAC Key

Client Write Key

Server Write Key

Client Write IV

Server Write IV
What’s the difference?

PRE-SHARED KEYS AND SESSIONS
Why do we need sessions?

• Full handshakes are expensive.
  – Key generation.
  – Server (& Client) Authentication.

• Many HTTP clients need it.
  – Download web page resources (JS, CSS, images).
  – Dynamic web pages (XHR).
  – May not be feasible to keep connection open.
How do we establish a PSK?

- **Out-of-band**
  - Added to TLS in 2006 via RFC4279.

- **During Handshake**
  - Client announces it supports session resumption.
  - Server provides a PSK *identities* during handshake.

- **After handshake, Server sends “New Session Ticket”**
  - Contains PSK identity, nonce and max age.
  - The PSK is derived from master secret.
  - Server can send multiple tickets.
So, TLS v1.3 supports PSK-based session resumption becomes...
What about Zero Round Trip Time (0-RTT)?

• PSK means the key is known to both sides.
  – Does this mean Client can send data immediately?
  – Can we have a zero round trip time handshake?

Yes, we can!

• But...
  – No forward secrecy for the “early data” sent by client.
  – No guarantees of non-replay.
So, TLS v1.3 supports PSK-based session resumption...
Extensions... Extensions everywhere!

BACKWARDS COMPATIBILITY
• Backwards compatibility is important
  – TLS v1.3 clients need to talk to TLS v1.2 servers.
  – TLS v1.2 clients need to talk to TLS v1.3 servers.

• Structure of Hello messages is maintained.
  – 12 extensions defined in the RFC.
  – 9 extensions defined in other RFCs.

• E.g. server key exchange message replaced with key_share extension.
All the extensions

<table>
<thead>
<tr>
<th>Extension</th>
<th>TLS 1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name [RFC6066]</td>
<td>CH, EE</td>
</tr>
<tr>
<td>max_fragment_length [RFC6066]</td>
<td>CH, EE</td>
</tr>
<tr>
<td>status_request [RFC6066]</td>
<td>CH, CR, CT</td>
</tr>
<tr>
<td>supported_groups [RFC7919]</td>
<td>CH, EE</td>
</tr>
<tr>
<td>signature_algorithms [RFC5246]</td>
<td>CH, CR</td>
</tr>
<tr>
<td>use_srtt [RFC5764]</td>
<td>CH, EE</td>
</tr>
<tr>
<td>heartbeat [RFC6520]</td>
<td>CH, EE</td>
</tr>
<tr>
<td>application_layer_protocol_negotiation [RFC7301]</td>
<td>CH, CR, CT</td>
</tr>
<tr>
<td>signed_certificate_timestamp [RFC6962]</td>
<td>CH, EE</td>
</tr>
<tr>
<td>client_certificate_type [RFC7250]</td>
<td>CH, EE</td>
</tr>
<tr>
<td>server_certificate_type [RFC7250]</td>
<td>CH, CT</td>
</tr>
<tr>
<td>padding [RFC7685]</td>
<td>CH</td>
</tr>
<tr>
<td>key_share</td>
<td>CH, SH, HRR</td>
</tr>
<tr>
<td>pre_shared_key</td>
<td>CH, SH</td>
</tr>
<tr>
<td>psk_key_exchange_modes</td>
<td>CH</td>
</tr>
<tr>
<td>early_data</td>
<td>CH, EE, NST</td>
</tr>
<tr>
<td>cookie</td>
<td>CH, HRR</td>
</tr>
<tr>
<td>supported_versions</td>
<td>CH</td>
</tr>
<tr>
<td>certificateAuthorities</td>
<td>CH, CR</td>
</tr>
<tr>
<td>oid_filters</td>
<td>CR</td>
</tr>
<tr>
<td>post_handshake_auth</td>
<td>CH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH</td>
<td>Client Hello</td>
</tr>
<tr>
<td>SH</td>
<td>Server Hello</td>
</tr>
<tr>
<td>EE</td>
<td>Encrypted Extensions</td>
</tr>
<tr>
<td>CT</td>
<td>Certificate</td>
</tr>
<tr>
<td>CR</td>
<td>Certificate Request</td>
</tr>
<tr>
<td>NST</td>
<td>New Session Ticket</td>
</tr>
<tr>
<td>HRR</td>
<td>Hello Retry Request</td>
</tr>
</tbody>
</table>
Backwards Compatibility Considerations

- Protocol Version is mentioned in every message.
  - Now deprecated/fixed to old version values
  - Handshake claims 1.2, App Data claims 1.0.
  - New extension specifies list of supported versions.

- Fixed values to prevent downgrade attacks.
  - Server “Random” has fixed last 8 bytes
    - DOWNGRD[0x01] for TLS 1.2 clients.
    - DOWNGRD[0x00] for <= TLS 1.1 clients.
• Removed
  – Anything that was unused, unsafe or didn’t offer significant value.

• Added
  – Handshake encryption.
  – 1-RTT and 0-RTT PSK / Session Resumption.

• Changed
  – Cipher Suites.
  – Handshake.
  – Hashed-Key Derivation Function (HKDF).
  – Key Schedule.
  – Sessions.
THANK YOU FOR LISTENING!
• The Good:
  – Massive efficiency gains*.
  – Fewer choices for Client & Server means reduced attack vectors.

• The Bad:
  – “Extensions.... extensions everywhere” (21)
  – A lot of added complexity for backwards compatibility.
  – Specification consumability is questionable.

* 0-RTT has a “whiff of future regret” about it.
Unused Slides

APPENDIX
What’s the point of the master secret?

• Client and Server need:
  – Keys for symmetric encryption.
  – Initialisation Vectors for AEAD Cipher Modes.

• Keys & IVs generated from a master secret.

• TLS defines a “Key Schedule”
  – How HKDF algorithm is used.
  – How to generate an infinite amount of secure key material.

• So, how does HKDF work?
HMAC-based Extract-and-Expand Key Derivation Function

HMAC (IS THE NEW PRF)
What is HKDF used for?

- **Key Schedules**
  - Handshake Secrets.
  - Early Traffic Secrets.
  - Master Secret.
  - Application Data Secrets.
  - Initialisation Vectors.

- **Transcript Hashes**
  - Certificate Verification.
  - Handshake “Finished” Keys.
• TLS <= v1.2 defines PRF algorithm.
  – HKDF encapsulates how TLS uses HMAC.
  – Re-used in other protocols.
  – Separate cryptographic analysis already done.

• Provides 2 functions:
  – **Extract** - create a pseudo-random key from inputs.
  – **Expand** - create more keys from the first key.

• **HMAC** is integral to HKDF.
• It creates a **Message Authentication Code** using:
  
  – Message data.
  
  – A shared key.
  
  – A cryptographic hash algorithm (set in cipher suite).
    
    • SHA256 or SHA384.

\[
\text{HMAC}(K, m) = H((K' \oplus \text{opad}) || H((K' \oplus \text{ipad}) || m))
\]
• Keyed-Hash Message Authentication Code

\[
\begin{align*}
0x5c5c5c5c5c5c5c5c\ldots & \quad \text{XOR'd Secret Key} \\
0x3636363636363636\ldots & \quad \text{message} \\
\text{hash} & \quad \text{hash} \\
\text{hash} & \quad \text{HMAC}
\end{align*}
\]
• Extract
  – Creates a Pseudo-Random Key (PRK)

\[
\text{HKDF-Extract}(\text{salt}, \text{IKM}) \rightarrow \text{PRK} \\
\text{PRK} = \text{HMAC-Hash}(\text{salt}, \text{IKM})
\]

Expand
  – Creates infinite key material from the PRK.
  – Iteratively calls HMAC with an increasing counter.

\[
\text{HKDF-Expand}(\text{PRK}, \text{info}, L) \rightarrow \text{OKM} \\
T(0) = \text{empty string (zero length)} \\
T(1) = \text{HMAC-Hash}(\text{PRK}, T(0) \mid \text{info} \mid 0x01) \\
T(2) = \text{HMAC-Hash}(\text{PRK}, T(1) \mid \text{info} \mid 0x02) \\
\text{...}
\]
However, it’s unfortunately not that simple...

\[
\text{Derive-Secret(Secret, Label, Messages[])} = \\
\text{HKDF-Expand(}
\text{Secret,}
\]

```plaintext

Length "tls13" Label HashValue
```

\[
\text{Hash(}
\text{Messages[0]} \text{ Messages[1]} \ldots \text{Messages[n]})
\]

Hash.Length
<table>
<thead>
<tr>
<th>CH Parameter</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol Version</td>
<td>Legacy slot for protocol version.</td>
<td>0x0303 TLS v1.2</td>
</tr>
<tr>
<td>Random</td>
<td>The Client Random</td>
<td>No more Unix time</td>
</tr>
<tr>
<td>Session ID</td>
<td>Session ID</td>
<td>Forced 0 byte length</td>
</tr>
<tr>
<td>Cipher Suites</td>
<td>Symmetric cipher options</td>
<td>One of Five</td>
</tr>
<tr>
<td>Compression Methods</td>
<td>N/A</td>
<td>Must specify not supported.</td>
</tr>
<tr>
<td>Supported Versions</td>
<td>List of uint16</td>
<td>0x0304 (TLS v1.3)</td>
</tr>
<tr>
<td>Signature Algorithms</td>
<td>List of supported</td>
<td>Required for Client Cert Auth</td>
</tr>
<tr>
<td>Negotiated Groups</td>
<td>Required for (EC)DHE</td>
<td></td>
</tr>
<tr>
<td>Key Share</td>
<td>Required for (EC)DHE</td>
<td></td>
</tr>
<tr>
<td>Pre-Shared Key</td>
<td>Required for PSK (incl. session resumption)</td>
<td></td>
</tr>
</tbody>
</table>
First Contact: Client Hello

- Client initiates the connection.
- Contents:
  - Version (Legacy)
    - Unused, must be set to 0x0303 (TLS v1.2)
  - Client Random
    - Used in PRF to create master secret.
  - Session ID (Legacy)
    - Ignored, kept for backwards compatibility.
  - Supported Cipher Suites
    - What cipher suites this client can support.
  - Compression (Legacy)
    - Ignored, kept for backwards compatibility
  - Extensions (TLS v1.3)
    - List of supported TLS versions (mandatory)
  - Extensions (Others)
    - Other extensions, e.g. SNI
• The problem with RSA key exchange:
  – The pre-master secret is always encrypted with the public certificate key in the certificate.
  – The certificate doesn’t change (often).
  – If the private key was ever compromised, Eve could read every conversation.
• Cryptographic hash algorithm features:
  – Find any \( m \) and \( m' \) such that \( \text{hash}(m)=\text{hash}(m') \)
  – Find \( m' \) given \( m \) such that \( \text{hash}(m)=\text{hash}(m') \)
  – Find \( m \) given \( x \) such that \( \text{hash}(m)=x \)

• MD5 vulnerabilities:
  – Collision attack – done.
  – Theoretical attack on pre-image (\( 2^{123} \) operations).

• SHA-1 vulnerabilities:
  – Collisions attack – given 6500 CPU-years or 1000-GPU years.
  – Reduced cryptographic strength from 160 bits to 77 bits.
Renegotiation Attacks [RRDO10]