TLS (mostly 1.3)

Thanks to Arne Ansper for most slides
Protocols in the TLS suite

- Handshake protocol
  - For agreeing on the symmetric keys
  - For checking the identity of the other party
    - Server is always authenticated. Client may or may not be authenticated
- Record protocol
  - For confidentiality and integrity of messages
- (Alert protocol)
SSL v2

- Server and optionally client authentication using X.509 certificates
- No proper key exchange, no PFS
- Session reuse
- RC4, RC2, IDEA, DES, 3DES with MD5
- 40 bit encryption for international usage
- Very brief and informal specification
- Development led by Taher ElGamal ?!
TLS 1.2

- Lots of options for everything – authentication, key exchange, session management
- Session renegotiation
- Large number of ciphersuites – stream ciphers, CBC block cipher, AEAD ciphers, national ciphers (e.g. GOST)
- Compression
- Lots of extensions – IANA has registered 47 extensions for TLS (about half of them look universally useful)
- Support for connectionless transport
TLS 1.3 or actually TLS 2

- Can co-exist with TLS 1.2 but is essentially a new design
- Simplified, unified, optionless, optimized, more secure
- Free from legacy algorithms
  - E.g. only five ciphersuites, all of them are AEAD algorithms
- Easier to analyse and prove the security properties
- Many cryptographic enhancements
- Cryptographic design based on Hugo Krawczuk's OPTLS
Supported algorithms

- **Authentication**
  - RSA (PKCS 1.5 and PSS), ECDSA (NIST curves), EdDSA (ed25519 and ed448), symmetric pre-shared keys (PSK)

- **Key-exchange**
  - DHE and ECDHE (NIST, x25519, x448)

- **Hash**
  - SHA256

- **Symmetric encryption algorithms**
  - AES 128/256 GCM/CBC and ChaCha20+Poly1305
    - SHA256/384 used for deriving the keys
AEAD – authenticated encryption with associated data

- A type of block ciphers’ mode of operation
- Inputs: key K, plaintext P, associated data A
  - In practice: also a nonce N that may not repeat
- Outputs: A, ciphertext C, tag T
- Confidentiality and integrity for P. Integrity for A
- In TLS 1.3:
  - AES128/256 with
    - CCM – Counter mode with CBC-MAC
    - GCM – Galois Counter Mode
  - ChaCha20 + Poly1305
    - Stream cipher + information-theoretic MAC
A polynomial evaluation based MAC

- Let $\mathbb{F}$ be a finite field of sufficient size (say, $\approx 2^{128}$)
- Let $\text{trunc} : \mathbb{F} \rightarrow \{0, 1\}^n$, where
  - $n$ is sufficiently large (say, $2^{128}$);
  - All sets $\text{trunc}^{-1}(m)$ (for $m \in \{0, 1\}^n$) are of size at most $S$
    - $2^n \approx S/|\mathbb{F}|$
- Let $\mathcal{N}$ be a set of nonces, $\mathcal{K}$ a set of keys and $\text{prf} : \mathcal{K} \times \mathcal{N} \rightarrow \{0, 1\}^n$ be pseudorandom.

The interface of the MAC

- The set of keys: $\mathbb{F}^* \times \mathcal{K}$
  - $\mathbb{F}^*$ — all invertible elements of $\mathbb{F}$, i.e. $\mathbb{F}\backslash\{0\}$
- The set of possible messages: $\bigcup_{\ell \in \mathbb{N}\backslash\{0\}} \mathbb{F}^\ell$
- The set of tags: $\mathcal{N} \times \{0, 1\}^n$
MAC computation

- Let \((r, k) \in \mathbb{F}^* \times \mathcal{K}\) be the key
- Let \(M = (M_1, M_2, \ldots, M_\ell)\) be the message.
- Randomly pick \(N \leftarrow \mathcal{N}\)
- Compute

\[
T = \text{trunc} \left( \sum_{i=1}^{\ell} M_i r^i \right) \oplus \text{prf}_k(N)
\]

- Output \((N, T)\)
- Verification: given \((r, k), M, N\), recompute \(T\)
MAC computation

Let \((r, k) \in \mathbb{F}^* \times \mathcal{K}\) be the key.
Let \(M = (M_1, M_2, \ldots, M_\ell)\) be the message.
Randomly pick \(N \leftarrow \mathcal{N}\).
Compute
\[
T = \text{trunc} \left( \sum_{i=1}^{\ell} M_i r^i \right) \oplus \text{prf}_k(N)
\]
Output \((N, T)\).
Verification: given \((r, k), M, N\), recompute \(T\).

A convenient notation

For given \(M = (M_1, M_2, \ldots, M_\ell)\) let \(P_M : \mathbb{F} \to \mathbb{F}\) be the polynomial
\[
P_M(x) = \sum_{i=1}^{\ell} M_i x^i
\]
Security

- MAC security: Existential unforgeability under Chosen Message Attack (EUF-CMA)
  - A key $K$ is generated. Adversary is given access to oracle $Mac_K(\cdot)$
  - Adversary must come up with a pair $(M, T)$ that it did not submit to the oracle, but which verifies wrt. the key $K$

- If $Mac$ is our polynomial evaluation based MAC, then...
  - Let adversary submit $M^{(1)}, \ldots, M^{(q)}$ with lengths $\ell^{(1)}, \ldots, \ell^{(q)}$
    - Let $(N^{(j)}, T^{(j)}) = Mac_{r,k}(M^{(j)})$
  - Adversary knows nothing about $P_{M^{(1)}}(r), \ldots, P_{M^{(q)}}(r)$
    - I.e. he knows nothing about $r$
  - Forging only works with an existing $N^{(j)}$
  - To forge, adversary has to pick $j$, a new message $M'$ of length $\ell'$, and a tag $T'$
Probability of successful forgery

\[ \text{trunc}(P_{M'}(r)) = T' \oplus \text{prf}_k(N^{(i)}) \]
Probability of successful forgery

\[ \text{trunc}(P_{M'}(r)) = T' \oplus \text{prf}_k(N^{(j)}) \]

\[ P_{M'}(r) \in \text{trunc}^{-1}(T' \oplus \text{prf}_k(N^{(j)})) \]
Probability of successful forgery

\[ \text{trunc}(P_{M'}(r)) = T' \oplus \text{prf}_k(N^{(i)}) \]

\[ P_{M'}(r) \in \text{trunc}^{-1}(T' \oplus \text{prf}_k(N^{(i)})) = \{v_1, \ldots, v_S\} \]
Probability of successful forgery

\[ \text{trunc}(P_{M'}(r)) = T' \oplus \text{prf}_k(N^{(j)}) \]

\[ P_{M'}(r) \in \text{trunc}^{-1}(T' \oplus \text{prf}_k(N^{(j)})) = \{v_1, \ldots, v_S\} \]

- The polynomial \( P_{M'}(x) - v_i \) has at most \( \ell' \) roots.
- Considering different \( i \)-s and different \( j \)-s, there are at most \( \ell'qS \) possible values of \( r \) that “work” for the given \( M' \).
- The probability of having one of these \( r \)-s is at most \( \ell'qS/|\mathbb{F}| \approx \ell'q \cdot 2^{-n} \).
Poly1305

- $\mathbb{F}$ is $GF(2^{130} - 5)$
- $M \in \{0, 1\}^*$ is split into 128-bit blocks (last block may be shorter). Each block is prepended with the bit 1
- Output length $n$ is 128. $\text{trunc}$ is $\text{lsb}_{128}$
- Original paper defined $\text{prf}$ as AES128. In TLS 1.3, ChaCha20 is used.
AES-GCM

- **Inputs:** $K$, $P$, $A$
- Get $IV$ from internal state. Compute $C = AES-CTR(K, IV, M)$
- Apply polynomial evaluation based MAC to $A \parallel C \parallel \ell_A \parallel \ell_C$, where
  - $F$ is $GF(2^{128})$. Output length is 128. $trunc$ is identity
  - The nonce is $IV$
    - Counter-mode encryption starts from $IV + 1$
- **Output** $C$ and the computed tag $T$
  - $IV$ is not part of the output
TLS record protocol

- Input: stream of messages
  - Each message has a “type”: application_data, handshake, change_cipher_spec, alert
- Messages are turned into records of at most 16K bytes
  - Fragmenting or coalescing generally OK
- Internal state — per-record nonce (64 bits)
  - One for each direction
  - Incremented for each record
  - Gives the IV
- Records are formatted, padded (if desired), encrypted with AEAD scheme
  - AD: length of C and some legacy stuff
Client

ClientHello
+ key_share*
+ signature_algorithms*
+ psk_key_exchange_modes*
+ pre_shared_key*

Unencrypted

Encrypted

{Certificate*}
{CertificateVerify*}
{Finished}

[Application Data]

Server

ServerHello
+ key_share*
+ pre_shared_key*

{EncryptedExtensions}
{CertificateRequest*}

{Certificate*}
{CertificateVerify*}
{Finished}

<-------- [Application Data*]

<--------> [Application Data]

<--------> [Application Data]
$C \rightarrow S : N_C, g^x$
$S \rightarrow C : N_S, g^y$
$S \rightarrow C : Cert_S$
$S \rightarrow C : [\{N_C, g^x, N_S, g^y, Cert_S\}]_{K_S}$
$S \rightarrow C : MAC_{g^{xy}}(N_C, g^x, N_S, g^y, Cert_S, sig_S)$
$C \rightarrow S : Cert_C$
$C \rightarrow S : [\{N_C, g^x, N_S, g^y, Cert_S, sig_S, mac_S, Cert_C\}]_{K_C}$
$C \rightarrow S : MAC_{g^{xy}}(N_C, g^x, N_S, g^y, Cert_S, sig_S, mac_S, Cert_C, sig_C)$
Client

ClientHello
+ key_share

Server

HelloRetryRequest
+ key_share

ClientHello
+ key_share

ServerHello
+ key_share

Unencrypted

Encrypted

{EncryptedExtensions}
{CertificateRequest*}

{Certificate*}
{CertificateVerify*}
{Finished}

[Application Data*]

{Certificate*}

{CertificateVerify*}

{Finished}

[Application Data]
Client Hello
+ key_share*
+ pre_shared_key

--------->

Server Hello
+ pre_shared_key
+ key_share*

{EncryptedExtensions}
{Finished}

[Application Data*]

<-------->

{Finished}

[Application Data]

[Application Data]
$C \rightarrow S : N_C, g^x$

$C \rightarrow S : \{(pskId_i, HMAC_{psk_i}(N_C, g^x, pskId_1, \ldots, pskId_n))\}_{i=1}^n$

$S \rightarrow C : N_S, g^y, pskId_k$

$S \rightarrow C : MAC_{H(psk_k, g^{xy})}(N_C, g^x, N_S, g^y, [psk]_C, pskId_k)$

$C \rightarrow S : MAC_{H(psk_k, g^{xy})}(N_C, g^x, N_S, g^y, [psk]_C, pskId_k, mac_S)$
\( C \rightarrow S : N_C \)
\( C \rightarrow S : \{ (pskId_i, \text{HMAC}_{psk_i} (N_c, g^x, pskId_1, \ldots, pskId_n)) \}_{i=1}^n \)
\( S \rightarrow C : N_S, pskId_k \)
\( S \rightarrow C : \text{MAC}_{psk_k} (N_C, N_S, [psk]_C, pskId_k) \)
\( C \rightarrow S : \text{MAC}_{psk_k} (N_C, N_S, [psk]_C, pskId_k, mac_S) \)
Client

ClientHello
  + key_share*
  + signature_algorithms*
  + psk_key_exchange_modes*
  + pre_shared_key*

Server

ServerHello
  + key_share*
  + pre_shared_key*

{EncryptedExtensions}
{CertificateRequest*}
{Certificate*}
{CertificateVerify*}
{Finished}

<--------  [Application Data*]

{Certificate*}
{CertificateVerify*}
{Finished}

<--------  [NewSessionTicket]

[Application Data]      <------->  [Application Data]
Client

ClientHello
+ early_data
+ key_share*
+ psk_key_exchange_modes
+ pre_shared_key

Encrypted (Application Data*) -------->

ServerHello
+ pre_shared_key
+ key_share*

{EncryptedExtensions}
+ early_data*

{Finished}

<-------- [Application Data*]

(EndOfEarlyData)

{Finished} -------->

[Application Data]

[Application Data] <-------->

Server
0-RTT data or early data

- Saves one round trip – getting response from server requires two round trips instead of three
- Can be used only with PSK
- 0-RTT data is not forward secret, as it is encrypted solely under keys derived using the offered PSK
- There are no guarantees of non-replay between connections
- 0-RTT data cannot be duplicated within a connection and an attacker will not be able to make 0-RTT data appear to be 1-RTT data
TLS 1.3 for IoT

- TLS 1.3 can be configured to work without using public key cryptography
- PSK authentication without DHE – only symmetric encryption and hash functions are needed – can be implemented on tiny microcontrollers
- Provably secure protocol – better than ad-hoc solutions
- No forward secrecy
Small features

- Client authentication can be performed after handshake is completed and some data is exchanged.
- OCSP responses and signed certificate timestamps can be sent together with the certificates to facilitate the certificate validation without making additional requests to external services.
PSK $\rightarrow$ HKDF-Extract = Early Secret
  $\rightarrow$ Derive-Secret("ext binder" | "res binder", ") = binder_key
  $\rightarrow$ Derive-Secret("c e traffic", ClientHello) = client_early_traffic_secret
  $\rightarrow$ Derive-Secret("e exp master", ClientHello) = early_exporter_master_secret

Derive-Secret("derived", "")

(EC)DHE $\rightarrow$ HKDF-Extract = Handshake Secret
  $\rightarrow$ Derive-Secret("c hs traffic", ClientHello...ServerHello) = client_handshake_traffic_secret
  $\rightarrow$ Derive-Secret("s hs traffic", ClientHello...ServerHello) = server_handshake_traffic_secret

Derive-Secret("derived", "")

0 $\rightarrow$ HKDF-Extract = Master Secret
  $\rightarrow$ Derive-Secret("c ap traffic", ClientHello...server Finished) = client_application_traffic_secret0
  $\rightarrow$ Derive-Secret("s ap traffic", ClientHello...server Finished) = server_application_traffic_secret0
  $\rightarrow$ Derive-Secret("exp master", ClientHello...server Finished) = exporter_master_secret
  $\rightarrow$ Derive-Secret("res master", ClientHello...client Finished) = resumption_master_secret