# Overview of the TLS Handshake

Christopher Dickerman

05/01/24

## Overview of Presentation

- Theoretical Primitives
  - Discrete Log Problem (DLP)
  - Hidden Subgroup Problem (HSP)
  - Finite Field Diffie-Hellman (DH)
  - Elliptic Curve Diffie-Hellman (ECDH)
- Implementation
  - Certificates and Establishment of Trust
  - RSA Digital Signature
  - Example TLS Handshake

## Discrete Log Problem (DLP)

Consider the group $(\mathrm{G}, \cdot)$ with order $N$ and elements $\{e, g, g^2, g^3, ..., g^{N-1}\}$, where $g^k = \underbrace{g \cdot g \cdot ... \cdot g}_{k \text{ factors}}$

**The Discrete Log Problem**: Given some $x \in \mathrm{G}$, find the smallest $a$ such that $g^a = x$. This is denoted

$$a = \log_g(x)$$

The logarithm is only unique $\mathrm{mod}(N)$, since for any positive integer $k$,

$$g^{a+kN} = g^a \cdot (g^N)^k = g^a \cdot e = g^a$$

## Discrete Log Problem (DLP)

- Easy $\rightarrow$: Given $g$ and $a$, calculate $g^a = x$
  - Naively $O(N)$, calculate $g^a$ by repeatedly multiplying $g$
  - Better $O(\log(N))$, Precompute $S = \{g, g^2, g^4, g^8, ..., g^{2^k}\}$, then $g^a$ is a sum of a subset of the elements of $S$
- Hard $\leftarrow$: Given $g$ and $x$, calculate $\log_g x = a$
  - Naively $O(N)$, calculate $g^k$ iterating $k$ until we get $x$
  - Best achieved is $O(\sqrt{N})$, e.g. "Baby-step giant-step"

## Hidden Subgroup Problem (HSP)

- Prime factorization and the DLP are instances of the HSP
- Quantum algorithms can solve the HSP in polynomial time

**The Hidden Subgroup Problem**: Given a group $\mathrm{G}$, a subgroup $\mathrm{H} \leq \mathrm{G}$, and a set $\mathrm{X}$, we say a function $f : \mathrm{G} \to \mathrm{X}$ **hides** the subgroup $\mathrm{H}$ if

$$f(g_1) = f(g_2) \iff g_1\mathrm{H} = g_2\mathrm{H}, \forall g_1, g_2 \in \mathrm{G}$$

Assume $f$ is an oracle and we don't know $\mathrm{H}$. The problem is to determine a generating set for $\mathrm{H}$ by using information gained by querying $f$.

## Hidden Subgroup Problem (HSP)

Example:

- $\mathrm{G} = \mathbb{Z}_4, \mathrm{H} = \langle 2 \rangle, \mathrm{X} = \mathbb{Z}_4/\langle 2 \rangle$
- $f : \mathbb{Z}_4 \to \mathbb{Z}_4/\langle 2 \rangle$ where $f(x) = x + \{0, 2\}$
- We don't know anything besides $\mathrm{G}$, and we want to determine whether $\mathrm{H}$ is $\langle 2 \rangle$ or $\{0\}$.
- To solve, we can just naively check every element and see if the coset is equal to $f(0)$. If $f(0) = f(a)$, we know that $a\mathrm{H} = 0\mathrm{H} \implies a \in H$
  - $f(0) = 0 + \langle 2 \rangle = \{0, 2\}$
  - $f(1) = 1 + \langle 2 \rangle = \{1, 3\}$
  - $f(2) = 2 + \langle 2 \rangle = \{2, 0\} = \{0, 2\}$
  - $f(3) = 3 + \langle 2 \rangle = \{3, 1\} = \{1, 3\}$

## DLP as an Instance of HSP

Let $\mathrm{G} = \langle g \rangle, |\mathrm{G}| = N$. Given $x \in \mathrm{G}$, we want to obtain $\log_g(x)$.

We can represent this as a HSP in the group $\mathbb{Z}_N \times \mathbb{Z}_N$

Let $f : \mathbb{Z}_N \times \mathbb{Z}_N \to \mathrm{G}$, $f(a,b) = x^a g^b$

$$f(a,b) = x^a g^b = g^{a \log_g(x) + b}$$

Notice that $f$ is constant along lines

$$L_c = \{(a,b) : a \log_g(x) + b = c \mod N\}$$

Each $L_c$ here is unique, that is if $c \neq c'$, then $L_c \neq L_{c'}$.

# DLP as an Instance of HSP

$$f(a,b) = x^a g^b = g^{a\log_g(x)+b}$$

Here's what the sets $L_c$ look like:

$$L_c = \{(0,c),(1,-\log_g(x)+c),(2,-2\log_g(x)+c),\ldots,(N-1,-(N-1)\log_g(x)+c)\}$$

And a set of particular interest to us is $L_0$:

$$L_0 = \{(0,0),(1,-\log_g(x)),(2,-2\log_g(x)),\ldots,(N-1,-(N-1)\log_g(x))\}$$

$\forall (a_0,b_0) \in L_0,\ f(a_0,b_0) = g^0 = 0$

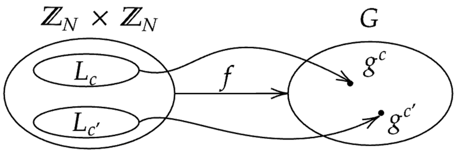


Figure: A graphical representation of $f : \mathbb{Z}_N \times \mathbb{Z}_N \to \mathrm{G}$

## DLP as an Instance of HSP

We can show that $f$ is a homomorphism:

$$\begin{aligned} f((a_1, b_1) \oplus (a_2, b_2)) &= f(a_1 + a_2, b_1 + b_2) \\ &= x^{a_1+a_2} g^{b_1+b_2} \\ &= x^{a_1} g^{b_1} x^{a_2} g^{b_2} \\ &= f(a_1, b_1) f(a_2, b_2) \end{aligned}$$

Notice that when $a \log_g(x) + b = 0$, $f(a, b) = 0$, so $\ker f = L_0$, and

$$\frac{\mathbb{Z}_N \times \mathbb{Z}_N}{L_0} \cong G$$

by the Fundamental Theorem on Homomorphisms

## DLP as an Instance of HSP

Given $\frac{\mathbb{Z}_N \times \mathbb{Z}_N}{L_0} \cong G$ and $f(a_1, b_1) = f(a_2, b_2)$, we have

$$(a_1, b_1)L_0 = (a_2, b_2)L_0$$

$$\begin{tikzcd}
\mathbb{Z}_n \times \mathbb{Z}_n \arrow[r, "f"] \arrow[d, "\pi"'] & G \\
\frac{Z_n \times Z_n}{\ker f} \arrow[ur, "\phi"'] &
\end{tikzcd}$$

Thus the construction yields a subgroup that reveals $\log_g(x)$ and satisfies the conditions of the Hidden Subgroup Problem.

## Finite Field Diffie-Hellman (DH)

Alice and Bob want to share a secret without an eavesdropper Eve being able to see that secret. Alice chooses a large prime $p$ and a number $g \in \mathbb{F}^*{}_p$. She sends this to Bob.
Then

- Alice:
  - Chooses a secret $a \in \mathbb{F}^*{}_p$
  - Computes $A = g^a$ and sends it to Bob
  - Receives $B$ and computes $B^a = (g^b)^a = g^{ab}$
- Bob:
  - Chooses a secret $b \in \mathbb{F}^*{}_p$
  - Computes $B = g^b$ and sends it to Alice
  - Receives $A$ and computes $A^b = (g^a)^b = g^{ab}$
- Eve:
  - Receives $A = g^a$
  - Receives $B = g^b$
  - Attempts to compute $g^{ab}$

## Elliptic Curve Diffie-Hellman (ECDH)

Elliptic curves are curves in $\mathbb{R}^2$ satisfying the equation

$$y^2 = x^3 + Ax + B$$

We can define a notion of "addition" on the points of a elliptic curve for points $P$ and $Q$. To compute $P \oplus Q$, create the line $PQ$. If $PQ$ intersects the curve on a point not $P$ or $Q$, call that point $R$. Then $P \oplus Q = -R$, otherwise $P \oplus Q = O$, where $O$ is a point at infinity, and also the identity element.

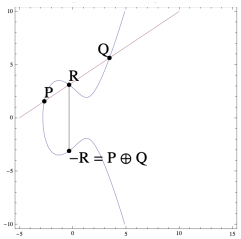


Figure: Elliptic curve "addition".

## Elliptic Curve Diffie-Hellman (ECDH)

This construction satisfies the following algebraic properties:

$$P \oplus Q = Q \oplus P$$
$$P \oplus O = P$$
$$P \oplus (Q \oplus R) = (P \oplus Q) \oplus R$$
$$P \oplus (-P) = O$$

And it gives us the following special rules:

If $P \neq Q$ and $x_1 = x_2$: $P \oplus Q = O$

If $P \neq Q$ and $y_1 = y_2$: $P \oplus Q = O$

Otherwise: $P \oplus Q = (m^2 - x_1 - x_2, -m^3 + m(x_1 + x_2) - b)$

With $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $m = \frac{y_2 - y_1}{x_2 - x_1}$ and $b = y_1 - mx_1$

## Elliptic Curve Diffie-Hellman (ECDH)

$$P \oplus Q = (m^2 - x_1 - x_2, -m^3 + m(x_1 + x_2) - b)$$

$$m = \frac{y_2 - y_1}{x_2 - x_1} \text{ and } b = y_1 - mx_1$$

When calculating on a finite field $\mathbb{F}^*{}_p$, we need to find $\frac{1}{x_2 - x_1}$ to calculate $m$. Fractions aren't allowed but given a value $a$ we can find $a^{-1}$. By Fermat's Little Theorem:

$$a^{p-1} \equiv 1 \bmod (p)$$

So $a^{-1} = a^{p-2}$

## Elliptic Curve Diffie-Hellman (ECDH)

In the group of $E(\mathbb{F}_p)$ of an elliptic curve over a finite field generated by $P$, we can define repeated addition:

$$n \cdot P = \underbrace{P \oplus P \oplus \cdots \oplus P}_{n \text{ times}}$$

**The Elliptic Curve Discrete Log Problem**: Given some $x \in E(\mathbb{F}_p)$, find the smallest $a$ such that

$$a \cdot P = x$$

## Elliptic Curve Diffie-Hellman (ECDH)

Alice and Bob want to share a secret without an eavesdropper Eve being able to see that secret. Alice chooses a large prime $p$ and a point $P \in E(\mathbb{F}^*{}_p)$. She sends this to Bob.
Then

- Alice:
  - Chooses a secret $a \in E(\mathbb{F}^*{}_p)$
  - Computes $A = a \cdot P$ and sends it to Bob
  - Receives $B$ and computes $a \cdot B = a \cdot (b \cdot P) = ab \cdot P$
- Bob:
  - Chooses a secret $b \in E(\mathbb{F}^*{}_p)$
  - Computes $B = b \cdot P$ and sends it to Alice
  - Receives $A$ and computes $b \cdot A = b \cdot (a \cdot P) = ab \cdot P$
- Eve:
  - Receives $A = a \cdot P$
  - Receives $B = b \cdot P$
  - Attempts to compute $ab \cdot P$

## TLS

TLS itself can be broken down into 2 parts, from the specification.

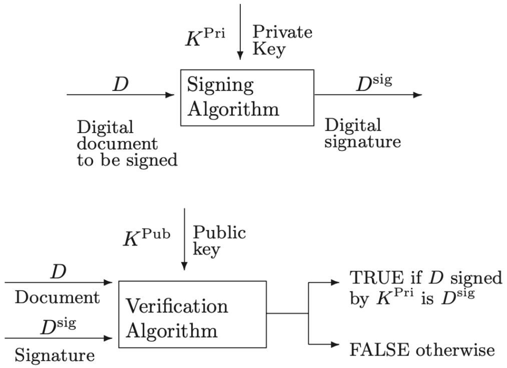
- "A handshake protocol that authenticates the communicating parties, negotiates cryptographic modes and parameters, and establishes shared keying material."
- "A record protocol that uses the parameters established by the handshake protocol to protect traffic between the communicating peers."

## TLS Handshake: Certificates

- I know I'm securely talking to someone, but how do I know who that is?
- MITM attack where Eve relays between Alice and Bob, supplying each with her public key $E$.
- Ideally we'd like some correspondence between real world entities and network entities that we can verify.

## Digital Signature Algorithms

Samantha wants to approve some document $D$ and provide information (signature) $D^{\text{sig}}$, so that given $D$ and $D^{\text{sig}}$, Victor can verify Samantha's approval. We can use the tools of asymmetric cryptography to do this.



## RSA Digital Signature Algorithm

**Euler's Formula for pq**: If $p$ and $q$ are distinct primes and $\gcd(a, pq) = 1$

$$a^{(p-1)(q-1)} \equiv 1 \mod pq$$

Proof: Start with showing that it is congruent modulo $p$

$$\left(a^{(p-1)}\right)^{(q-1)} \equiv 1^{(q-1)} \mod p$$

$$1^{(q-1)} \equiv 1 \mod p$$

Now showing that it is congruent modulo $q$

$$\left(a^{(q-1)}\right)^{(p-1)} \equiv 1^{(p-1)} \mod q$$

$$1^{(p-1)} \equiv 1 \mod q$$

## RSA Digital Signature Algorithm

$$\implies p \mid \left(a^{(p-1)(q-1)} - 1\right) \text{ and } q \mid \left(a^{(p-1)(q-1)} - 1\right)$$

$$\implies pq \mid \left(a^{(p-1)(q-1)} - 1\right)$$

$$\implies a^{(p-1)(q-1)} \equiv 1 \mod pq$$

If we add $1$ to the exponent then we get

$$a^{(p-1)(q-1)+1} \equiv a \mod pq$$

If we have some $s$ such that $\gcd(s,(p-1)(q-1)) = 1$, then there is an inverse which is to say there is some $v$ and $k$ such that

$$sv \equiv 1 \mod (p-1)(q-1)$$

$$\implies a^{sv} \equiv a^{1+k(p-1)(q-1)} \equiv a \mod pq$$

## RSA Digital Signature Algorithm

$$sv \equiv 1 \mod (p-1)(q-1)$$

Now if we have some document $D$ we want to authenticate, we can sign it with

$$S \equiv D^s \mod pq$$

And call $S$ the signature, then we can verify this signature with $v$ and get the document back

$$S^v \equiv D^{sv} \equiv D \mod pq$$

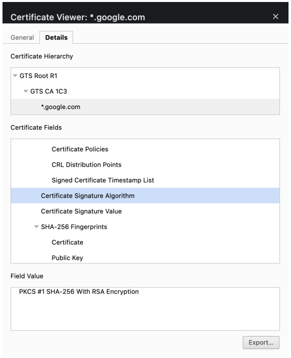
## RSA Digital Signature Algorithm

Victor wants to make sure that a document $D$ has been signed by Samantha and that it hasn't been forged. Samantha chooses two large primes $p$ and $q$ with $pq = N$ and some verification exponent with $\gcd(v, (p-1)(q-1)) = 1$. She publishes $(N, v)$. Then she calculates $s$ that satisfies $sv \equiv 1 \mod ((p-1)(q-1))$

- Samantha:
    - Calculates $S \equiv D^s \mod (pq)$ and sends it to Victor
- Victor:
    - Receives $S$ and computes $S^v \equiv D^{sv} \equiv D \mod N$

## TLS Handshake: Certificates

We can use the RSA signature algorithm to make sure a site is who they say they are. Looking at google.com shows us they are using PKCS #1 SHA-256 With RSA Encryption



Certificate Hierarchy

- GTS Root R1
  - GTS CA 1C3
    - *.google.com

Certificate Fields

- Certificate Policies
- CRL Distribution Points
- Signed Certificate Timestamp List
- Certificate Signature Algorithm
- Certificate Signature Value
- SHA-256 Fingerprints
  - Certificate
  - Public Key

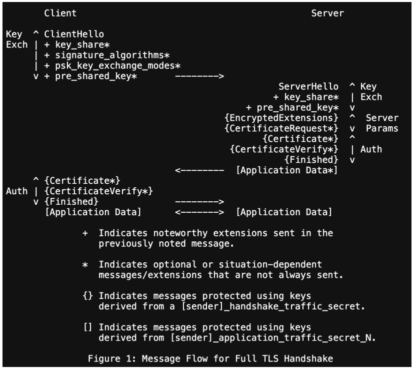
Field Value

PKCS #1 SHA-256 With RSA Encryption

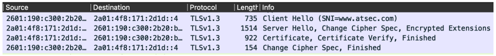
- Root certificates are stored in browser
- Certificate authorities can grant further certificates
- This builds a hierarchy of trust

# TLS Handshake

Now that we know who we are talking to, we can begin the TLS handshake.



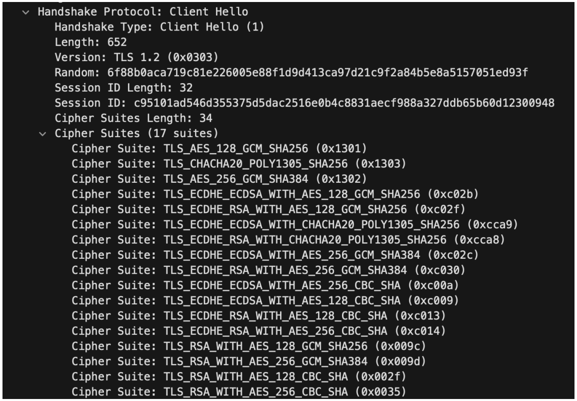
```
       Client                                           Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | + psk_key_exchange_modes*
     v + pre_shared_key*       -------->
                                                  ServerHello  ^ Key
                                                 + key_share*  | Exch
                                            + pre_shared_key*  v
                                        {EncryptedExtensions}  ^  Server
                                        {CertificateRequest*}  v  Params
                                               {Certificate*}  ^
                                         {CertificateVerify*}  | Auth
                                                   {Finished}  v
                               <--------  [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}              -------->
       [Application Data]      <------->  [Application Data]

              +  Indicates noteworthy extensions sent in the
                 previously noted message.

              *  Indicates optional or situation-dependent
                 messages/extensions that are not always sent.

              {} Indicates messages protected using keys
                 derived from a [sender]_handshake_traffic_secret.

              [] Indicates messages protected using keys
                 derived from [sender]_application_traffic_secret_N.

               Figure 1: Message Flow for Full TLS Handshake
```



| Source | Destination | Protocol | Lengt | Info |
|---|---|---|---|---|
| 2601:190:c300:2b20… | 2a01:4f8:171:2d1d::4 | TLSv1.3 | 735 | Client Hello (SNI=www.atsec.com) |
| 2a01:4f8:171:2d1d:… | 2601:190:c300:2b20:b… | TLSv1.3 | 1514 | Server Hello, Change Cipher Spec, Encrypted Extensions |
| 2a01:4f8:171:2d1d:… | 2601:190:c300:2b20:b… | TLSv1.3 | 922 | Certificate, Certificate Verify, Finished |
| 2601:190:c300:2b20… | 2a01:4f8:171:2d1d::4 | TLSv1.3 | 154 | Change Cipher Spec, Finished |

# TLS Handshake: clientHello

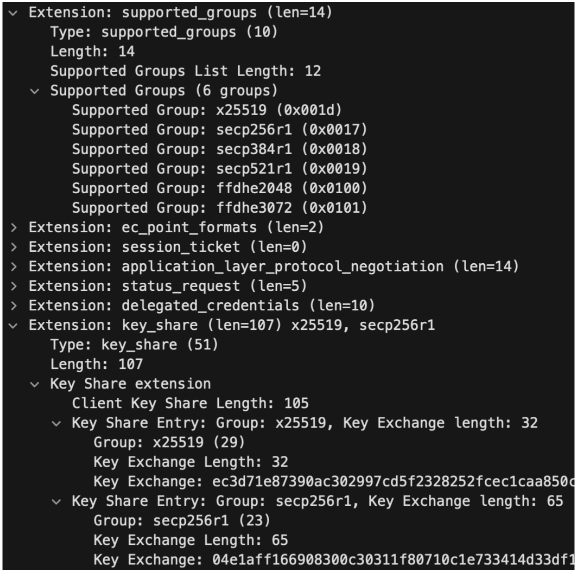
We begin with the `clientHello`



```
⌄ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 652
    Version: TLS 1.2 (0x0303)
    Random: 6f88b0aca719c81e226005e88f1d9d413ca97d21c9f2a84b5e8a5157051ed93f
    Session ID Length: 32
    Session ID: c95101ad546d355375d5dac2516e0b4c8831aecf988a327ddb65b60d12300948
    Cipher Suites Length: 34
  ⌄ Cipher Suites (17 suites)
      Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
      Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
      Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
      Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
      Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
      Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

## TLS Handshake: clientHello

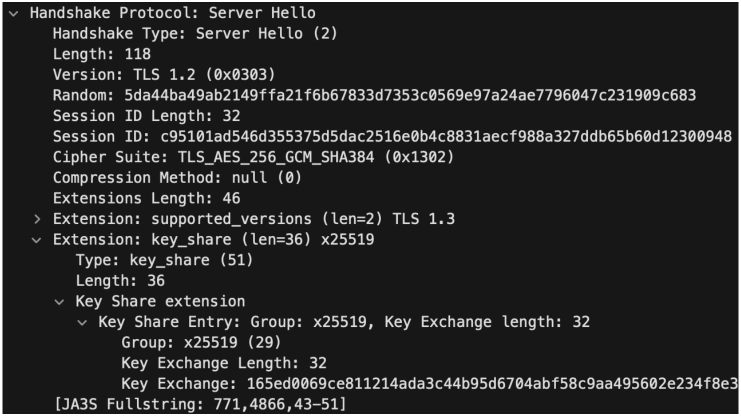
The `clientHello` also includes the supported groups for DH or ECDH, and preemptively includes some public keys.



```
⌄ Extension: supported_groups (len=14)
    Type: supported_groups (10)
    Length: 14
    Supported Groups List Length: 12
  ⌄ Supported Groups (6 groups)
      Supported Group: x25519 (0x001d)
      Supported Group: secp256r1 (0x0017)
      Supported Group: secp384r1 (0x0018)
      Supported Group: secp521r1 (0x0019)
      Supported Group: ffdhe2048 (0x0100)
      Supported Group: ffdhe3072 (0x0101)
› Extension: ec_point_formats (len=2)
› Extension: session_ticket (len=0)
› Extension: application_layer_protocol_negotiation (len=14)
› Extension: status_request (len=5)
› Extension: delegated_credentials (len=10)
⌄ Extension: key_share (len=107) x25519, secp256r1
    Type: key_share (51)
    Length: 107
  ⌄ Key Share extension
      Client Key Share Length: 105
    ⌄ Key Share Entry: Group: x25519, Key Exchange length: 32
        Group: x25519 (29)
        Key Exchange Length: 32
        Key Exchange: ec3d71e87390ac302997cd5f2328252fcec1caa850c
    ⌄ Key Share Entry: Group: secp256r1, Key Exchange length: 65
        Group: secp256r1 (23)
        Key Exchange Length: 65
        Key Exchange: 04e1aff166908300c30311f80710c1e733414d33df1
```

## TLS Handshake: serverHello

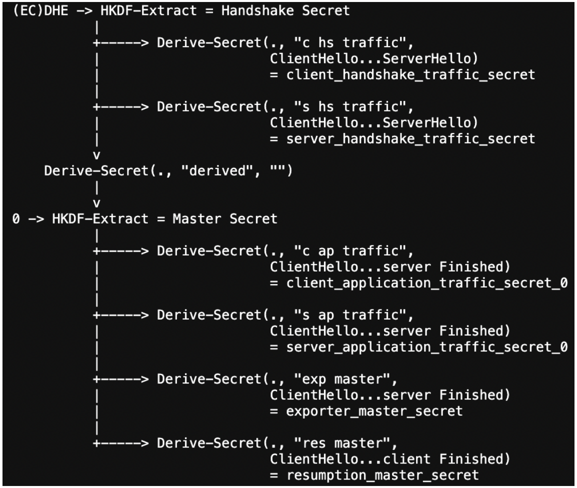
The `serverHello` responds with the selected cipher suite and their public ECDH x25519 public key.



```
⌄ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 118
    Version: TLS 1.2 (0x0303)
    Random: 5da44ba49ab2149ffa21f6b67833d7353c0569e97a24ae7796047c231909c683
    Session ID Length: 32
    Session ID: c95101ad546d355375d5dac2516e0b4c8831aecf988a327ddb65b60d12300948
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Compression Method: null (0)
    Extensions Length: 46
  › Extension: supported_versions (len=2) TLS 1.3
  ⌄ Extension: key_share (len=36) x25519
      Type: key_share (51)
      Length: 36
    ⌄ Key Share extension
      ⌄ Key Share Entry: Group: x25519, Key Exchange length: 32
          Group: x25519 (29)
          Key Exchange Length: 32
          Key Exchange: 165ed0069ce811214ada3c44b95d6704abf58c9aa495602e234f8e3
    [JA3S Fullstring: 771,4866,43-51]
```

## TLS Handshake: Key Schedule

Once we have the shared secret established through ECDH on the group x25519, we use this to derive a master key.



```
(EC)DHE -> HKDF-Extract = Handshake Secret
           |
           +-----> Derive-Secret(., "c hs traffic",
           |                     ClientHello...ServerHello)
           |                     = client_handshake_traffic_secret
           |
           +-----> Derive-Secret(., "s hs traffic",
           |                     ClientHello...ServerHello)
           |                     = server_handshake_traffic_secret
           v
    Derive-Secret(., "derived", "")
           |
           v
0 -> HKDF-Extract = Master Secret
           |
           +-----> Derive-Secret(., "c ap traffic",
           |                     ClientHello...server Finished)
           |                     = client_application_traffic_secret_0
           |
           +-----> Derive-Secret(., "s ap traffic",
           |                     ClientHello...server Finished)
           |                     = server_application_traffic_secret_0
           |
           +-----> Derive-Secret(., "exp master",
           |                     ClientHello...server Finished)
           |                     = exporter_master_secret
           |
           +-----> Derive-Secret(., "res master",
                                 ClientHello...client Finished)
                                 = resumption_master_secret
```

## TLS Handshake: Hash Key Derivation Function HKDF

HKDF has two modules, HKDF-Extract, and HKDF-Expand.

- **HKDF-Extract** takes input key material and an optional salt, and outputs a pseudorandom key (PRK)
- **HKDF-Expand** takes the PRK from the last step, some "info", and a Length, and expands the pseudorandomness of the PRK to the desired length

HKDF depends on Hash based Message Authentication Code (HMAC) which is defined as the following:

$$\text{HMAC}(K, m) = H\left((K' \oplus opad) || H\left((K' \oplus ipad) || m\right)\right)$$

*Thank you*