Encrypted Client Hello

An all-in-one presentation

Georgios Tasopoulos (g.tasop@isi.gr)
Evangelos Haleplidis (haleplidis@isi.gr)
Apostolos Fournaris (fournaris@isi.gr)

TLS 1.3

TLS 1.3

- The de-facto standard for secure communications
- Client Hello initiates the handshake (illustrated TLS 1.3)
- Unfortunately some fields (extensions) remain unencrypted which contain sensitive information
  - Server Name Indication Extension (**SNI**)
  - Application-Layer Protocol Negotiation Extension (**ALPN**)

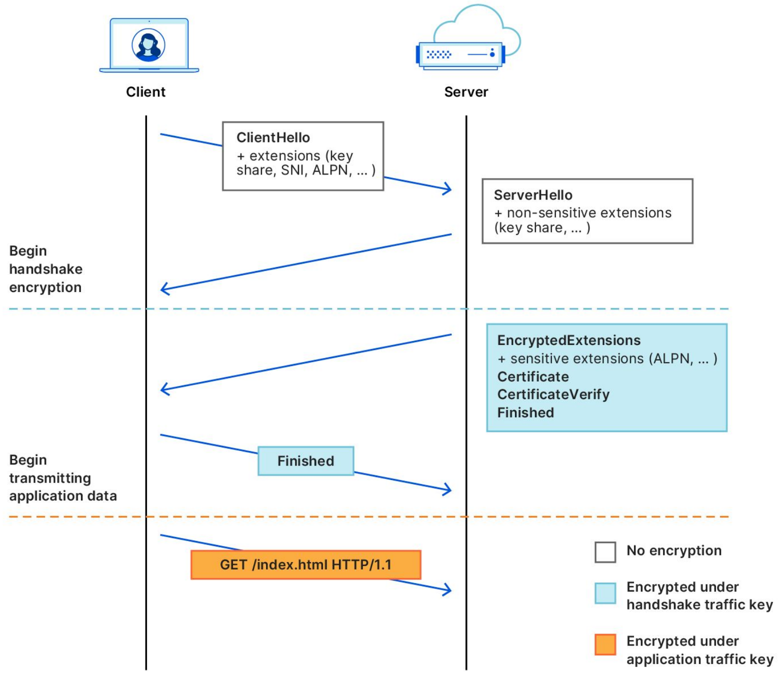
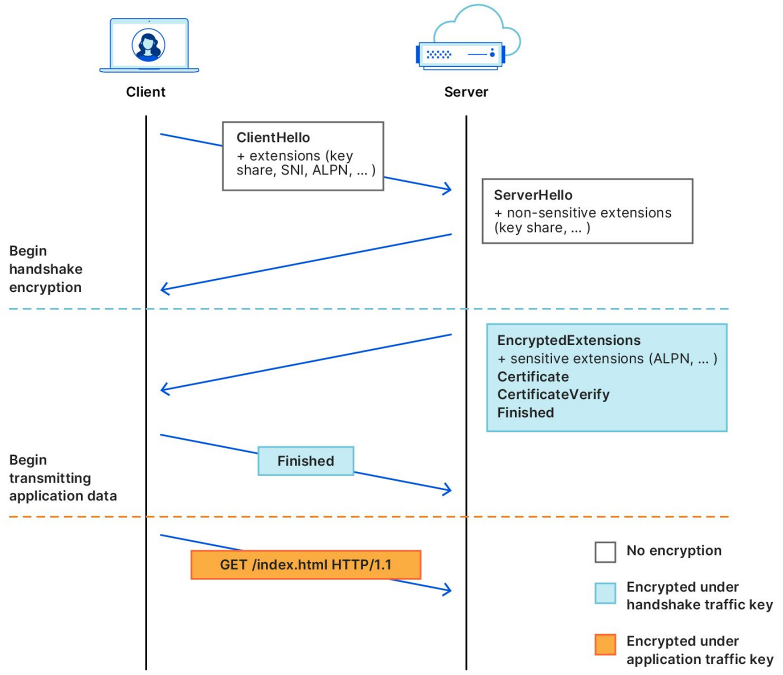

Client
Server
ClientHello
+ extensions (key share, SNI, ALPN, ...)
ServerHello
+ non-sensitive extensions (key share, ...)
Begin handshake encryption
EncryptedExtensions
+ sensitive extensions (ALPN, ...)
Certificate
CertificateVerify
Finished
Begin transmitting application data
Finished
GET /index.html HTTP/1.1
No encryption
Encrypted under handshake traffic key
Encrypted under application traffic key

Image source: Cloudflare (link)

ECH Extension

ECH Extension

- Sensitive unencrypted extensions
  - Server Name Indication Extension (**SNI**)
  - Application-Layer Protocol Negotiation Extension (**ALPN**)
- ECH (an IETF draft) is aiming to encrypt these extension before sending them to the server
- “*the goal of ECH is to encrypt all handshake parameters except those that are essential to completing the key exchange*”, Cloudflare blog bost
- The issue is...How is this achieved if the client sends the first message?



Client
Server
ClientHello + extensions (key share, SNI, ALPN, ...)
ServerHello + non-sensitive extensions (key share, ...)
Begin handshake encryption
EncryptedExtensions + sensitive extensions (ALPN, ...) Certificate CertificateVerify Finished
Finished
Begin transmitting application data
GET /index.html HTTP/1.1
No encryption
Encrypted under handshake traffic key
Encrypted under application traffic key

Image source: Cloudflare (link)

ECH Extension

- The answer: *DNS records*
- Together with the standard DNS query, the client requests for a TXT record with the ECH configuration (incl. Public key)
- This can be considered safe if DNS-over-HTTPS is used!
- It adds up to that and provides improvements to some known key distribution issues
- We are **NOT** going to use DNS in our examples; we will provide the client with fresh configs from the server assuming that it could get those from DNS records

ECH Extension - Client Hello

ECH has 2 Client Hello:

- *Client Hello Outer*
  - This is a full-fledged Client Hello message but with the bare minimum extensions and without any sensitive information
- *Client Hello Inner:*
  - This is the intended Client Hello message that the client want the server to use
- If the decryption with the Client Hello Inner *fails*, e.g outdated ECH config from the DNS record, the server continues with the Client Hello Outer only to provide the client with the correct ECH configuration and then the client aborts the connection!
- Now the client should retry connecting with the correct server-provided ECH configuration

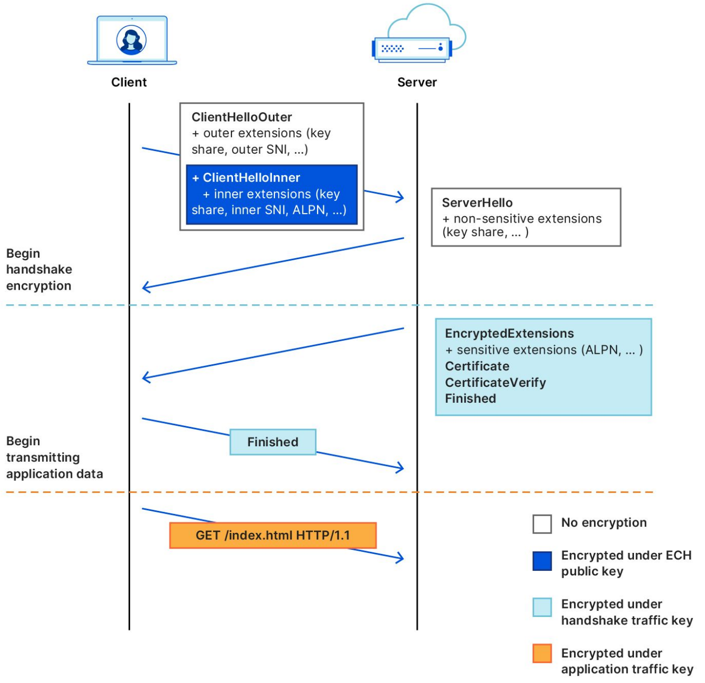

Client
Server
ClientHelloOuter
+ outer extensions (key share, outer SNI, ...)
+ ClientHelloInner
+ inner extensions (key share, inner SNI, ALPN, ...)
ServerHello
+ non-sensitive extensions (key share, ...)
Begin handshake encryption
EncryptedExtensions
+ sensitive extensions (ALPN, ...)
Certificate
CertificateVerify
Finished
Finished
Begin transmitting application data
GET /index.html HTTP/1.1
No encryption
Encrypted under ECH public key
Encrypted under handshake traffic key
Encrypted under application traffic key

Image source: Cloudflare (link)

ECH Extension - HPKE

- ECH uses the HPKE IETF draft for encrypting the sensitive extensions with a given public key.
- This draft "...*specifies an extensible framework for building public key encryption schemes suitable for a wide variety of applications*"
- And also "...*ECH delegates all of the details of its handshake encryption mechanism to HPKE, resulting in a much simpler and easier-to-analyze specification*"

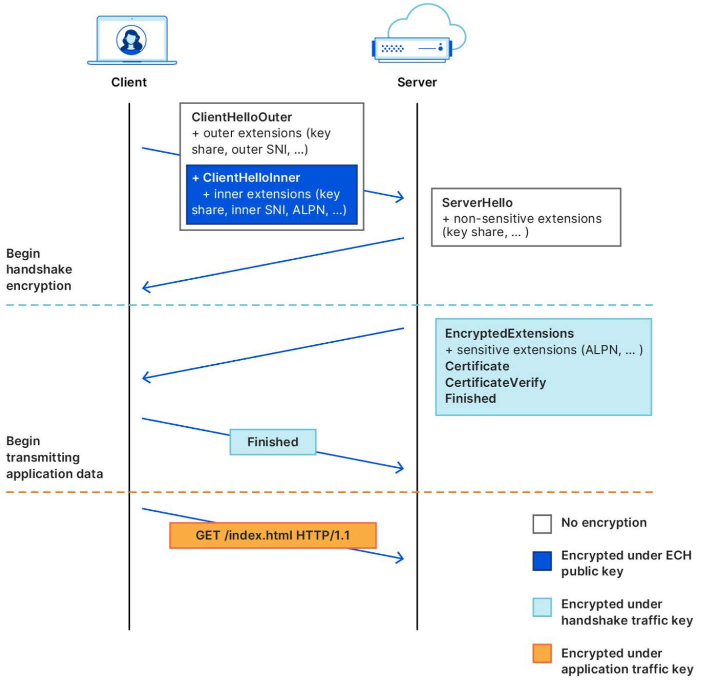

Client
Server
ClientHelloOuter
+ outer extensions (key share, outer SNI, ...)
+ ClientHelloInner
+ inner extensions (key share, inner SNI, ALPN, ...)
ServerHello
+ non-sensitive extensions (key share, ...)
Begin handshake encryption
EncryptedExtensions
+ sensitive extensions (ALPN, ...)
Certificate
CertificateVerify
Finished
Finished
Begin transmitting application data
GET /index.html HTTP/1.1
No encryption
Encrypted under ECH public key
Encrypted under handshake traffic key
Encrypted under application traffic key

Image source: Cloudflare (link)

ECH Extension - Future threats

- ECH key management solves a number of issues like cache-poisoning and general key management
- As it uses classical cryptography (as the rest of the TLS 1.3) ECH Extension is completely useless against an adversary in possession of a large-scale quantum computer.
- To mitigate this threat, we can drop-in post-quantum cryptography in the ECH extension
- A number of problems arise…

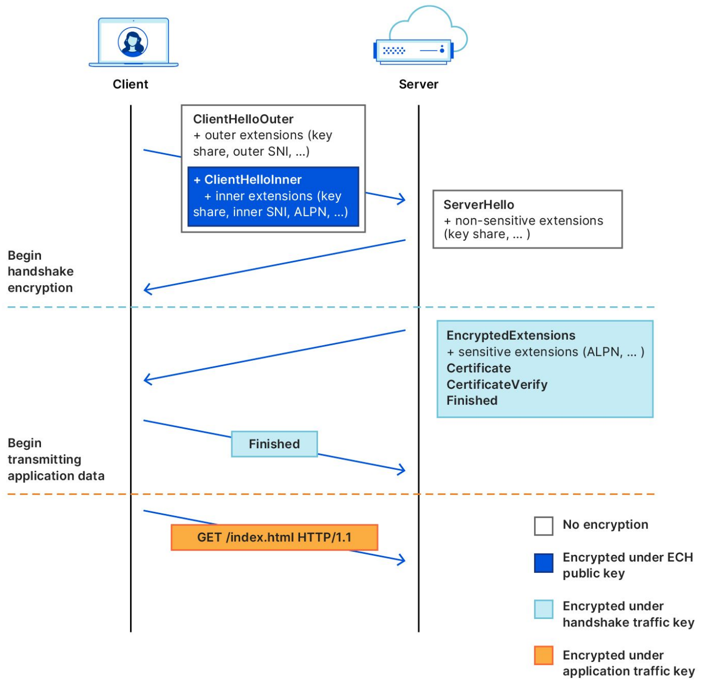

Client
Server
ClientHelloOuter
+ outer extensions (key share, outer SNI, ...)
+ ClientHelloInner
+ inner extensions (key share, inner SNI, ALPN, ...)
ServerHello
+ non-sensitive extensions (key share, ...)
Begin handshake encryption
EncryptedExtensions
+ sensitive extensions (ALPN, ...)
Certificate
CertificateVerify
Finished
Finished
Begin transmitting application data
GET /index.html HTTP/1.1
No encryption
Encrypted under ECH public key
Encrypted under handshake traffic key
Encrypted under application traffic key

Image source: Cloudflare (link)

Post-quantum cryptography

Post-quantum cryptography

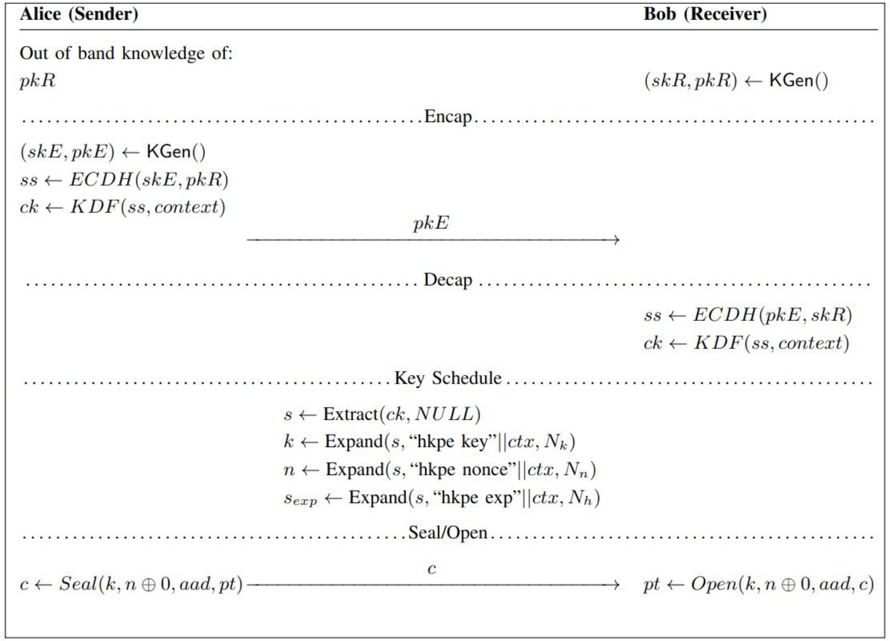
- In response to the threat of quantum attacks, **post-quantum cryptography** emerged
- NIST concluded the PQ process and 4 algorithms were selected for standardisation (3 standard drafts already published)
    - FIPS 203, *Module-Lattice-Based Key-Encapsulation Mechanism Standard*
    - FIPS 204, *Module-Lattice-Based Digital Signature Standard*
    - FIPS 205, *Stateless Hash-Based Digital Signature Standard*
- In our case we need a Key Encapsulation Mechanism (KEM) to replace the ECC curve algorithms
- Our only option is Kyber KEM - now known as ML-KEM

Post-quantum cryptography - ML-KEM

- Post-quantum KEMs (and KEMs in general) introduce an asymmetry in the usual public key operations with ECDH.
- All the details are presented in the work: PQ-HPKE: Post-Quantum Hybrid Public Key Encryption (eprint)
- It provides overviews of the classical HPKE and with the PQ-HPKE. Also it provides description and schemes of a hybrid HPKE, that uses PQ and ECDH keys.

Post-quantum cryptography - HPKE

Classical HPKE overview



Alice (Sender) **Bob (Receiver)**

Out of band knowledge of:

pkR $(skR, pkR) \leftarrow \mathsf{KGen}()$

Encap

$(skE, pkE) \leftarrow \mathsf{KGen}()$

$ss \leftarrow ECDH(skE, pkR)$

$ck \leftarrow KDF(ss, context)$

$\xrightarrow{pkE}$

Decap

$ss \leftarrow ECDH(pkE, skR)$

$ck \leftarrow KDF(ss, context)$

Key Schedule

$s \leftarrow \text{Extract}(ck, NULL)$

$k \leftarrow \text{Expand}(s, \text{"hkpe key"} || ctx, N_k)$

$n \leftarrow \text{Expand}(s, \text{"hkpe nonce"} || ctx, N_n)$

$s_{exp} \leftarrow \text{Expand}(s, \text{"hkpe exp"} || ctx, N_h)$

Seal/Open

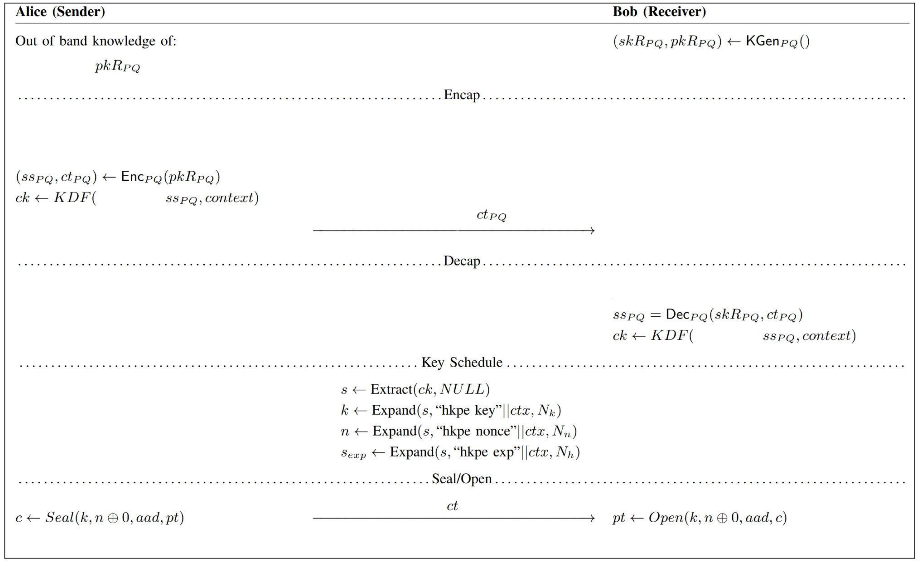
$c \leftarrow Seal(k, n \oplus 0, aad, pt) \xrightarrow{c} pt \leftarrow Open(k, n \oplus 0, aad, c)$

Fig. 1: HPKE Overview

Image source: https://eprint.iacr.org/2022/414.pdf

Post-quantum cryptography - HPKE

PQ-HPKE overview



Alice (Sender) | **Bob (Receiver)**

Out of band knowledge of: pkR_{PQ} | $(skR_{PQ}, pkR_{PQ}) \leftarrow \mathsf{KGen}_{PQ}()$

Encap

$(ss_{PQ}, ct_{PQ}) \leftarrow \mathsf{Enc}_{PQ}(pkR_{PQ})$

$ck \leftarrow KDF(\quad ss_{PQ}, context)$

$\xrightarrow{ct_{PQ}}$

Decap

$ss_{PQ} = \mathsf{Dec}_{PQ}(skR_{PQ}, ct_{PQ})$

$ck \leftarrow KDF(\quad ss_{PQ}, context)$

Key Schedule

$s \leftarrow \text{Extract}(ck, NULL)$

$k \leftarrow \text{Expand}(s, \text{“hkpe key”}||ctx, N_k)$

$n \leftarrow \text{Expand}(s, \text{“hkpe nonce”}||ctx, N_n)$

$s_{exp} \leftarrow \text{Expand}(s, \text{“hkpe exp”}||ctx, N_h)$

Seal/Open

$c \leftarrow Seal(k, n \oplus 0, aad, pt)$ $\xrightarrow{ct}$ $pt \leftarrow Open(k, n \oplus 0, aad, c)$

Fig. 2: PQ-only and PQ-hybrid HPKE Overview

Image source (edited): https://eprint.iacr.org/2022/414.pdf wolfSSL

wolfSSL

The implementation that we will use for “tweaking” the TLS 1.3 ECH extension into using PQ algorithms is ***wolfSSL***.

wolfSSL - Roadmap

We have documented the complete roadmap available here:

https://github.com/IETF-Hackathon/pq-ech/blob/main/documentation/roadmap.md