

Architecture TLS 1.3 et Handshake

TLS 1.3 est la version la plus récente du protocole Transport Layer Security, standardisée en 2018 (RFC 8446).



Le Principe

L'architecture TLS 1.3 s'organise en couches:

Application Layer ↓ TLS Record Protocol |—— Handshake Protocol |—— Alert Protocol |—— Change Cipher Spec (legacy) |—— Application Data Protocol ↓ Transport Layer (TCP)

TLS PROTOCOL



Composants clés de TLS 1.3

1

Record Protocol

Fragmente, compresse, chiffre et authentifie les données

3

Alert Protocol

Signale les erreurs et avertissements

2

Handshake Protocol

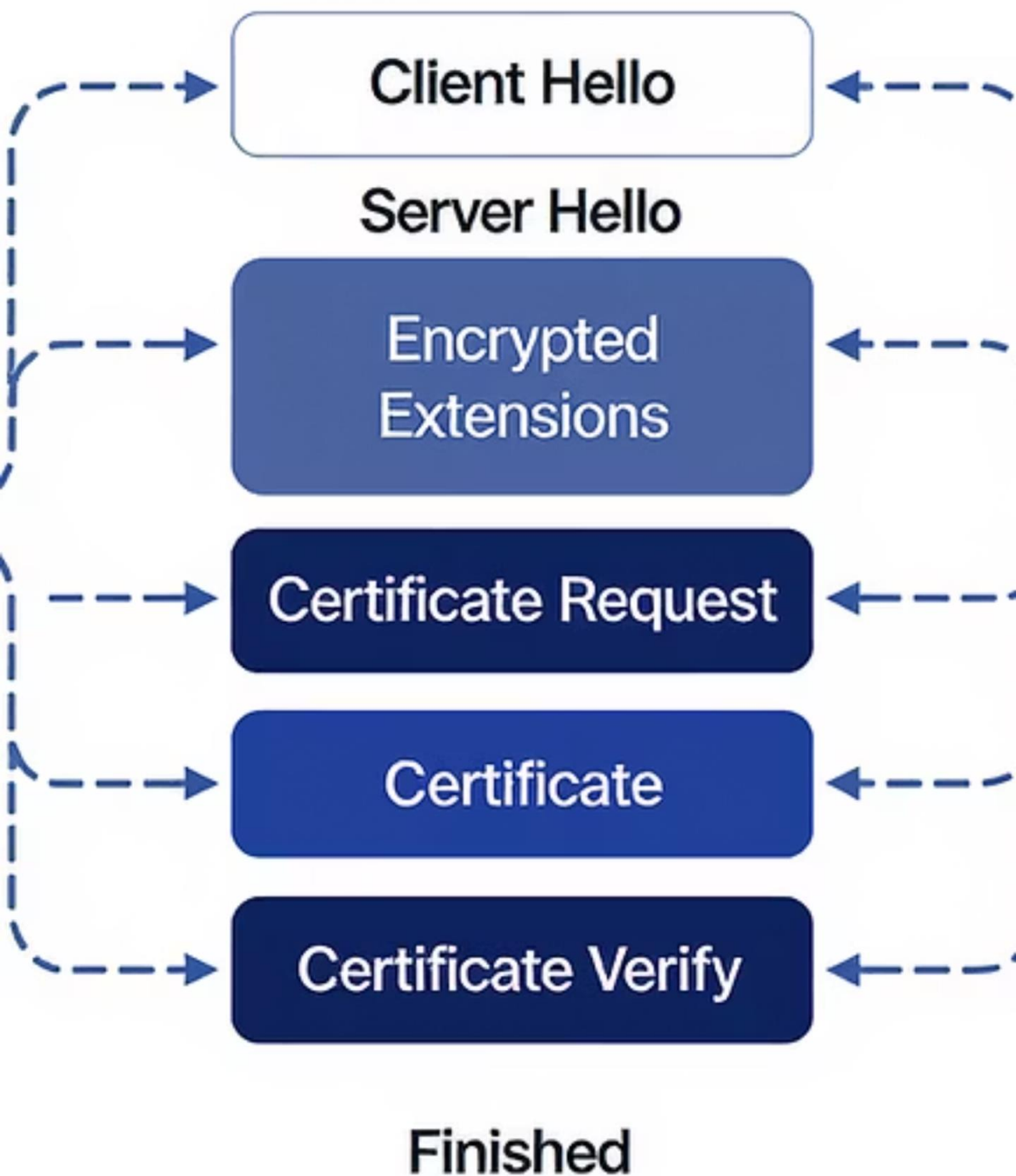
Négocie les paramètres de sécurité et établit les clés

4

Application Data Protocol

Transporte les données chiffrées

S 1.3 Handshak Sequence



Vue d'ensemble du Handshake TLS 1.3

```

Client Server||----- ClientHello ----->||
||<----- ServerHello -----||<-----
{EncryptedExtensions} -----||<----- {CertificateRequest*}
-----||<----- {Certificate} -----||<-----
{CertificateVerify} -----||<----- {Finished} -----
-----|| -----||-----
{Certificate*} ----->||----- {CertificateVerify*} -----
----->||----- {Finished} ----->||
||<===== [Application Data] =====>|
  
```

Légende : {} = chiffré, * = optionnel

ClientHello (1 RTT)

Le client envoie :

- Version TLS supportée : TLS 1.3
- Random : 32 octets aléatoires (protection contre le rejeu)
- Session ID : Compatibilité legacy
- Cipher Suites : Liste des algorithmes supportés

Extensions importantes :

- supported_versions : Indique TLS 1.3
- supported_groups : Courbes elliptiques (X25519, P-256...)
- key_share : Clés publiques éphémères pré-calculées
- signature_algorithms : RSA-PSS, ECDSA...
- server_name (SNI) : Nom du serveur

Cipher Suites TLS 1.3 (exemples) :- TLS_AES_128_GCM_SHA256- TLS_AES_256_GCM_SHA384- TLS_CHACHA20_POLY1305_SHA256

ServerHello et étapes suivantes

ServerHello

Le serveur répond avec :

- Version : TLS 1.3
- Random : 32 octets
- Cipher Suite sélectionnée
- key_share : Sa clé publique éphémère

Point clé : À ce stade, client et serveur peuvent déjà calculer les clés de chiffrement !

Encrypted Extensions

Premier message chiffré du serveur contenant :

- Extensions négociées
- Paramètres additionnels
- ALPN (protocole application)

Certificate et CertificateVerify

Le serveur envoie sa chaîne de certificats (format X.509v3) qui doit correspondre au SNI.

Puis il prouve la possession de la clé privée avec :

Signature = Sign(PrivKey, Transcript-Hash)
Transcript-Hash = Hash(ClientHello || ... || Certificate)

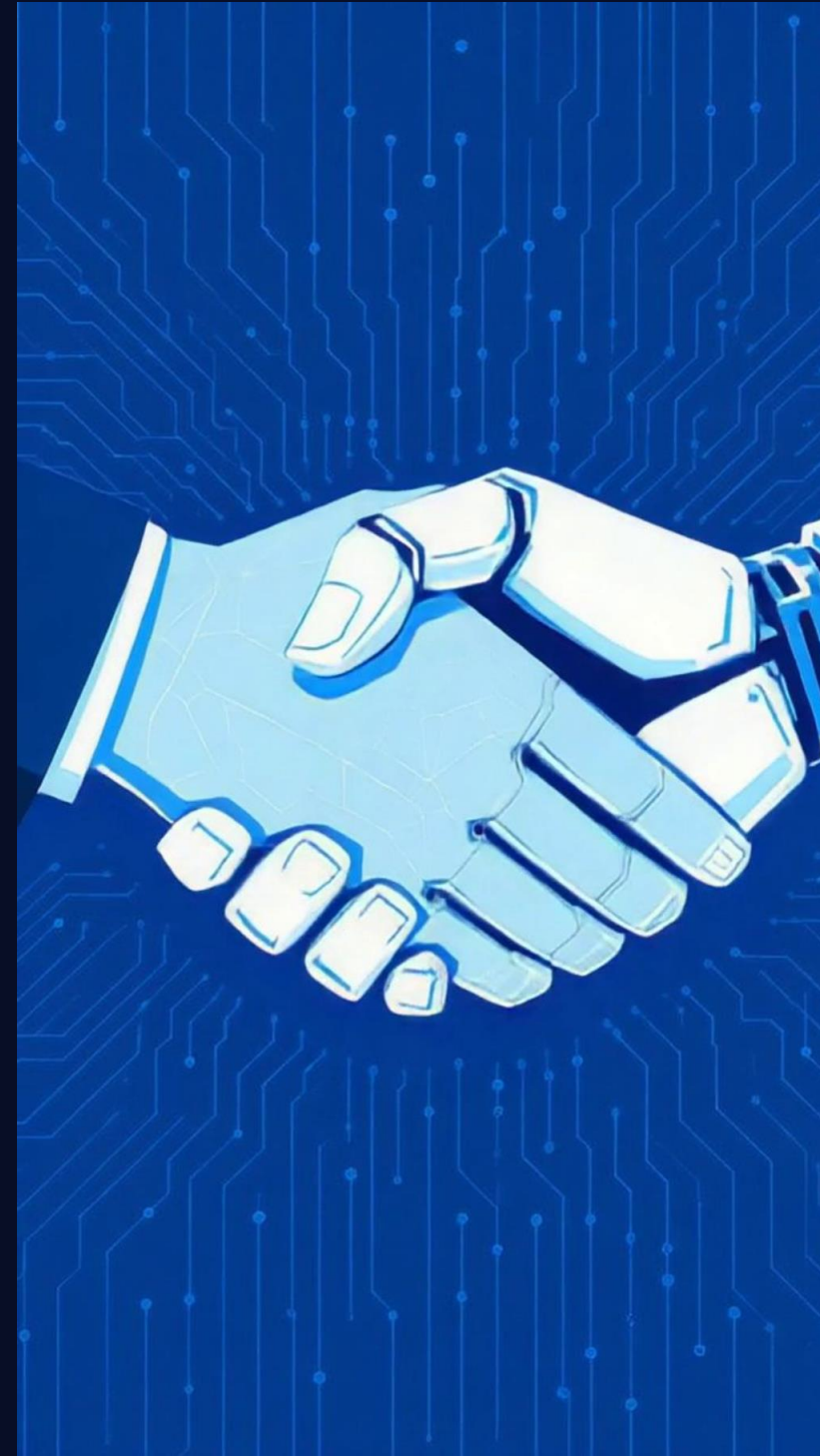
Finalisation du Handshake

Finished

MAC sur l'ensemble du handshake :

`finished_key = HKDF-Expand(BaseKey, "finished", "")`
`verify_data = HMAC(finished_key, Transcript-Hash)`

Ce message termine la phase d'établissement de la connexion sécurisée.

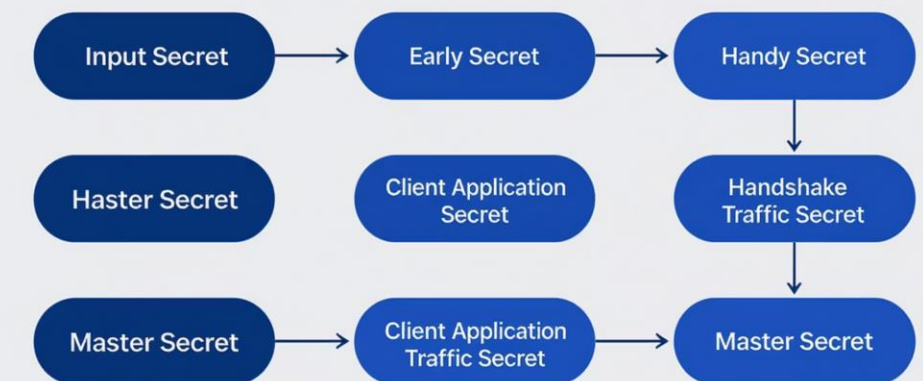


Génération des Clés (Key Schedule)

TLS 1.3 utilise HKDF (HMAC-based Key Derivation Function) :

```
0|vPSK -> HKDF-Extract = Early Secret|vDerive-Secret = binder_key|v(EC)DHE ->  
HKDF-Extract = Handshake Secret|+-----> Derive-Secret =  
client_handshake_traffic_secret|+-----> Derive-Secret =  
server_handshake_traffic_secret|vHKDF-Extract = Master Secret|+-----> Derive-  
Secret = client_application_traffic_secret|+-----> Derive-Secret =  
server_application_traffic_secret
```

TLS 1.3 Key Derivation Process



Améliorations de sécurité dans TLS 1.3

Suppression des algorithmes faibles

- Plus de RSA statique (vulnérable à l'absence de PFS)
- Plus de CBC mode
- Plus de compression
- Plus de renegotiation

Perfect Forward Secrecy (PFS) obligatoire

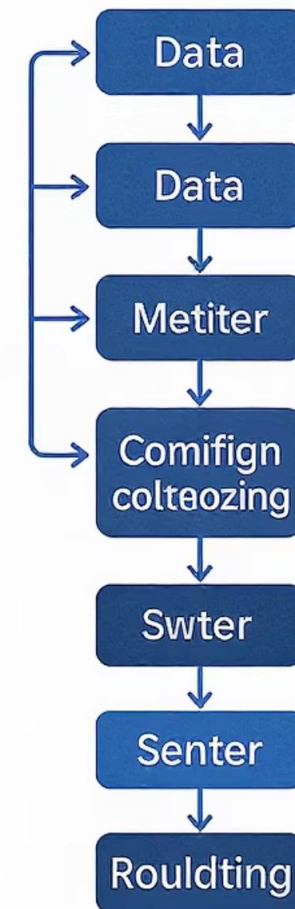
- Utilisation systématique de Diffie-Hellman éphémère
- Même si la clé privée du serveur est compromise, les sessions passées restent sécurisées

Améliorations de performance

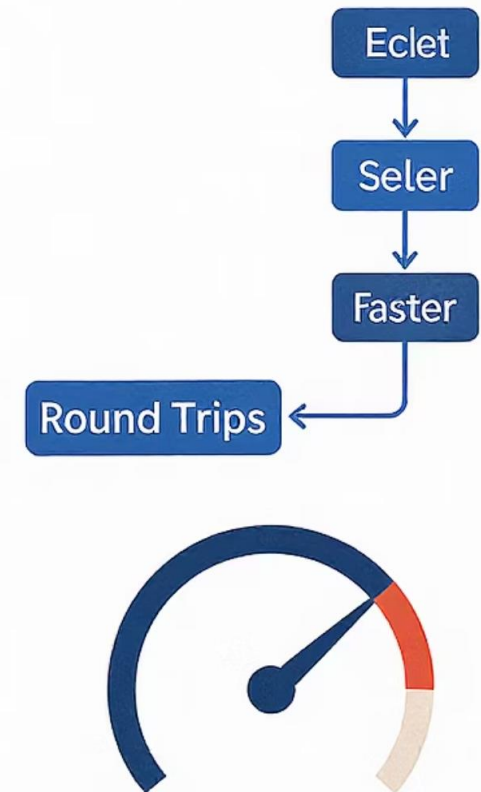
Performances améliorées

- 1-RTT Handshake (contre 2-RTT en TLS 1.2)
- 0-RTT Resumption : Réutilisation de session avec données immédiates
- Moins d'allers-retours : Chiffrement dès ServerHello

TLS 1.2 HANDSHAKE



VS TLS 1.3 Handshack Performance

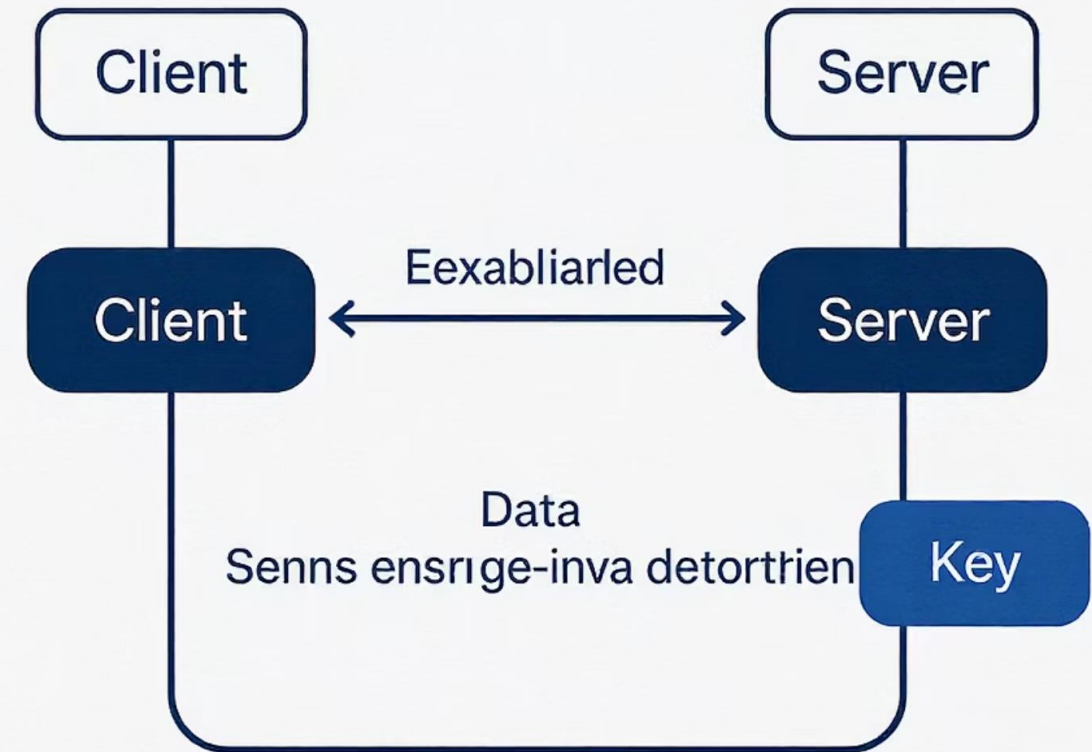


0-RTT Mode (Optionnel)

Client Server|---- ClientHello + 0-RTT Data ---->||<--- ServerHello + EncData -----||<===== 1-RTT Data =====>|

Attention : Vulnérable aux attaques par rejeu !

TLS 1.3 zero round trip time (0-RTT) Handshake



Exemple de Configuration Nginx

```
# Configuration TLS 1.3 modernssl_protocols TLSv1.3;ssl_ciphers
TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256;ssl_prefer_server_ciphers off;# OCSP
Staplingssl_stapling on;ssl_stapling_verify on;# Session tickets (pour 0-RTT)ssl_session_tickets
on;ssl_session_timeout 1d;ssl_session_cache shared:SSL:10m;# Early data (0-RTT) - à utiliser avec
précautionssl_early_data on;
```

Considérations de Sécurité



Menaces

1. Downgrade Attacks
2. Man-in-the-Middle
3. Replay Attacks
4. 0-RTT Replay



Protections

1. Protégé par supported_versions
2. Certificats + CertificateVerify
3. Random values + timestamps
4. Limiter aux requêtes idempotentes

Bonnes Pratiques



Certificats

Utiliser des certificats avec des clés d'au moins 2048 bits (RSA) ou 256 bits (ECC)



Renouvellement

Renouveler régulièrement les certificats



HSTS

Implémenter HSTS (HTTP Strict Transport Security)



Surveillance

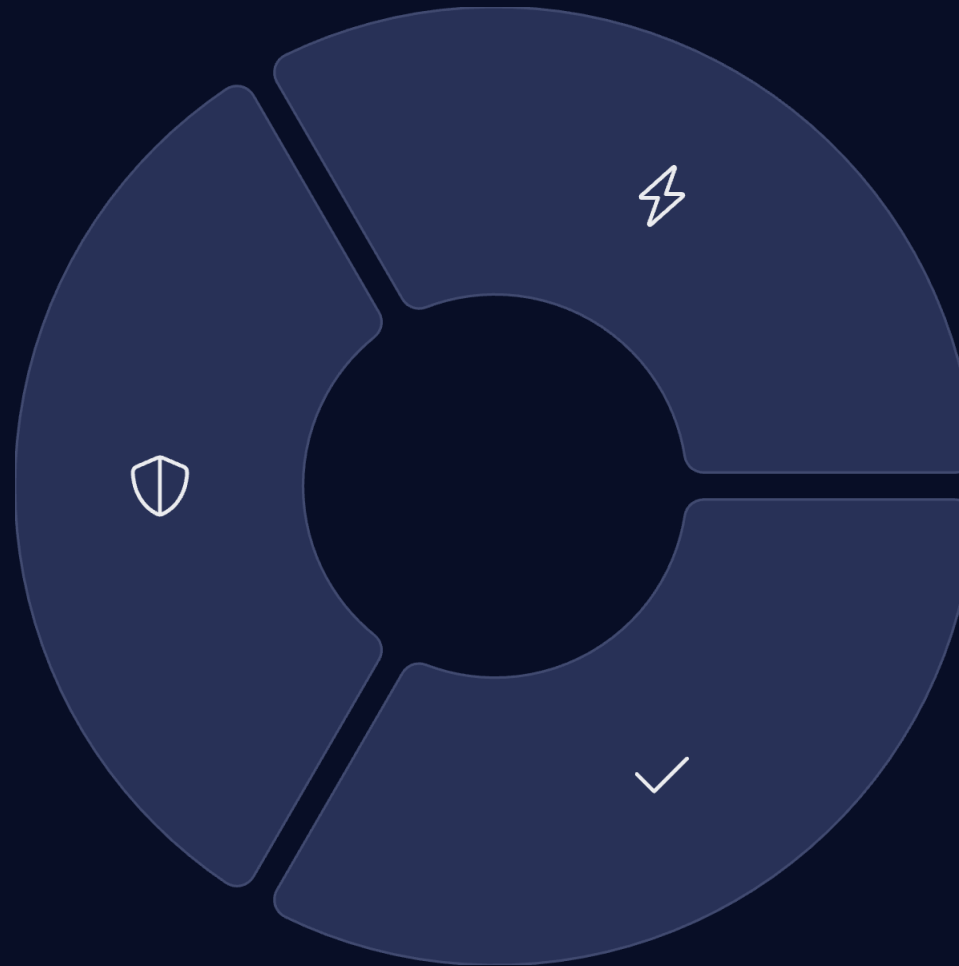
Surveiller les CT logs (Certificate Transparency)

Points Clés à Retenir

Sécurité + Performance

✓ TLS 1.3 = Sécurité + Performance

- Handshake en 1 RTT
- PFS obligatoire
- Algorithmes modernes uniquement



Chiffrement précoce

✓ Chiffrement précoce

- Dès le ServerHello
- Protection maximale du handshake

Simplicité

✓ Simplicité

- Moins d'options = moins d'erreurs
- Configuration plus sûre par défaut

Cette architecture représente l'état de l'art en matière de sécurisation des communications réseau, essentielle pour protéger les données en transit dans notre monde hyperconnecté.