Charles Olivier-Anclin
charles.olivier-anclin@uca.fr
perso.isima.fr/~chaolivi

Clermont Auvergne INP, Université Clermont Auvergne

Cours de Sécurité Web ZZ2F5

HTTPS : TLS, PKI

TLS - Transport Layer Security

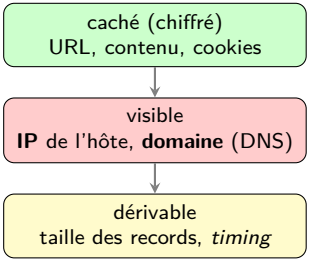
HTTP + SSL/TLS = HTTPS

- Confidentialité
- Intégrité
- Authentification (via les certificats)

Les format pour les suites cryptographiques sont :

Pour TLS 1.2 : TLS_KE_CIPHER_HASH
Pour TLS 1.3 : TLS_AENC-CIPHER_HASH

Visible *vs* caché dans le trafic



Anonymat

- Proxy http
- VPN
- Tor

Tester un certificat SSL/TLS

- SSL Certificate Decoder

Tester un serveur SSL/TLS

- Qualys
- testssl.sh
- `nmap -p 443 --script=ssl-enum-ciphers heart.bleed`

Tester un navigateur

- How's my SSL ?

What can go wrong ?

2015 Weak Diffie-Hellman and the Logjam Attack permet de forcé les connexions TLS à 512-bit
Freack Attack réminiscence

2014 **Poddle** :

- POODLE test

2014 Heartbleed : lecture de la mémoire du serveur via un heartbeat

D'autres : BEAST (weak CBC, SSL3/TLS1.0), HEIST (compression), DROWN (SSLv2), goto fail (erreur de code), et d'autres...

- OpenSSL vulnerabilities
- GnuTLS security
- Les attaques SSL / TLS
- Public Key Infrastructure (PKI)
  - Certificate authorities issue SSL certificates to fraudsters

Se protéger

- un service sans s est un problème
  - pas ftp, mais sftp ou ftps
  - pas rsync, mais rsync over ssh
  - pas imap, pop3 et smtp, mais imaps, pop3s et smtps
- Seules les implémentations conformes à TLSv2 et supérieures doivent être employées
- Les cyphersuites offrant la PFS (protection des sessions passés en cas de compromissions des clefs maitresses) doivent être favorisées
- Anssi - SSL/TLS : état des lieux et recommandations
- Transition vers le post-quantique

- Apache
  - Chiffrement fort SSL/TLS : Mode d'emploi différents services
- https://syslink.pl/cipherlist/
- fixer le weak Diffie-Hellman (aka logjam Attack)

```
openssl dhparam -out dhparams.pem 2048
openssl dhparam -out dhparams.pem -5 2048
```

  Ou utiliser des valeurs prédéfinis : RFC 7919
- suivre les recommandations de l'ANSSI

Section 2 : **Bases cryptographiques**

Objectifs de la cryptographie

Les quatre objectifs de sécurité *(vue 2.Priv)*

- **Confidentialité** : seul le destinataire légitime déchiffre
- **Intégrité** : le message n'a pas été modifié
- **Authenticité** : on connaît l'émetteur
- **Non-répudiation** : l'émetteur ne peut nier avoir envoyé

Liens avec TLS

TLS combine ces objectifs via des briques **symétriques**, **asymétriques** et des **fonctions de hachage**.

- **confidentialité** → chiffrement (symétrique, clés de session)
- **intégrité** → MAC / HMAC (hachage + clé)
- **authenticité du serveur** → signature / certificat X.509

Fonctions de hachage et MAC (rappel)

Hachage : $h(m)$

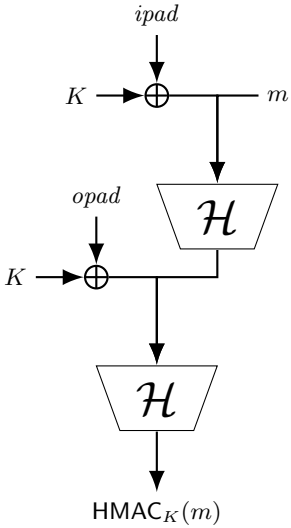
- empreinte de taille fixe, à sens unique
- pré-image, collision difficiles (SHA-256)
- n'apporte **ni confidentialité, ni authenticité**

MAC / HMAC → intégrité + authenticité

- $\text{MAC}(K, m) = H(K \parallel m)$ ou **HMAC** : $\text{HMAC}_K(m)$
- une **clé** partagée lie l'empreinte au message

En TLS, l'**intégrité** de chaque message est protégé par un **MAC** calculé avec la clé de session.

MAC basé sur une fonction de hachage (HMAC)

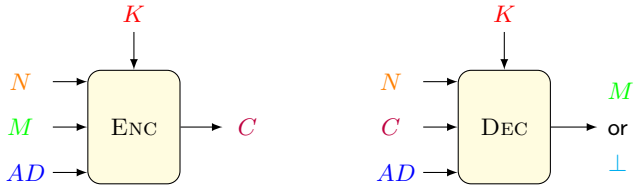


Avec $ipad = 0x36363636 \ldots 36$ et $opad = 0x5c5c5c5c \ldots 5c$.
Formalisé dans le RFC 2104

Intégrité du “handshake” (phase de négociation).

Authenticated Encryption with Associated Data

confidentialité et **intégrité** en un seul passage (le chiffrement seul ne détecte aucune modification) :



AEAD : GCM, ChaCha20-Poly1305 et TLS 1.3

Deux constructions

AES-GCM (RFC 5288)

- **CTR** (compteur) pour le chiffrement, **GHASH** (dans $\{0,1\}^{128}$) pour le MAC
- $c \,\|\, \tau = \mathrm{CTR}_{K_1}(m) \,\|\, \mathrm{GHASH}_{K_2}(\mathrm{hd} \,\|\, c \,\|\, \mathrm{len})$
- nonce unique : une réutilisation trahit K_2

ChaCha20-Poly1305 (RFC 8439)

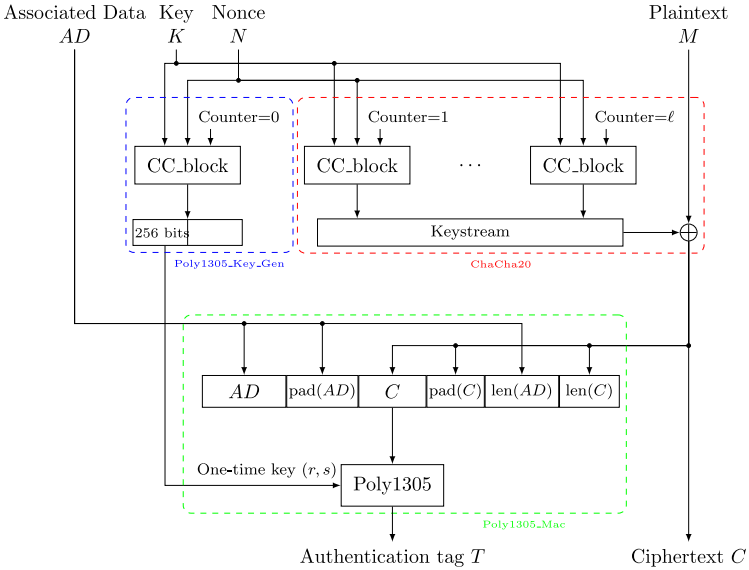
- flux **ChaCha20** (compteur + *nonce*) et MAC **Poly1305** à **usage unique**
- **sans accélération matérielle** (routeur, mobile), mais proches d'AES-GCM en logiciel

Dans le *record layer*

- hd = l'en-tête du record (`type` ‖ `version` ‖ `length`) : plus de *downgrade*
- plus de **clé MAC séparée** ni de **padding** ⇒ plus d'*oracle de remplissage*

TLS 1.3 **n'autorise que des suites AEAD** (…_GCM_SHA256, …_CHACHA20_POLY1305_SHA256).

Chacha20_poly1305



Section 3 : **TLS et X.509**

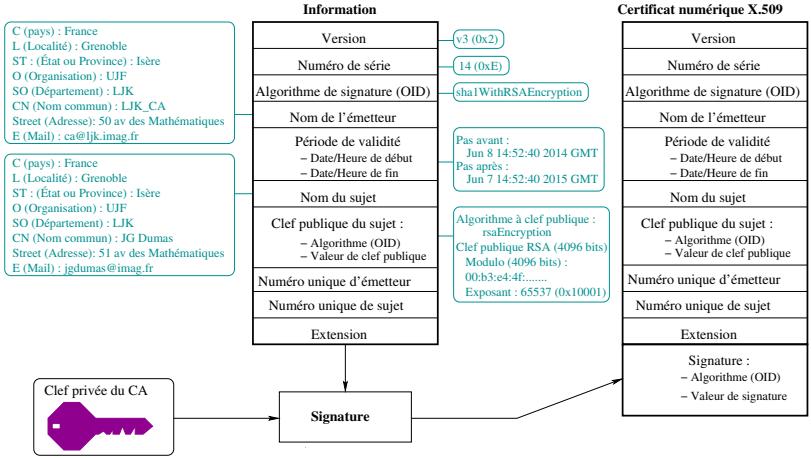
Norme de l'UIT (agence de l'ONU dédiée aux TIC)

- norme pour
  - les **certificats à clé publique**
  - les **listes de révocation de certificats** (CRL)
  - les **attributs** de certificat
  - l'algorithme de validation du **chemin de certification**
- versions : **v1** (1988), **v2** (1993), **v3** (1996)

Système hiérarchique d'autorités de certification

- une *Certification Authority* (**CA**) attribue un certificat liant une **clé publique** à un nom distinctif (*Distinguished Name*, **DN**)
- le **DN** composé d'attributs : `Country`, `Locality`, `STate`, `Organisation`, `OU`, `CN` (Common Name), `Street`, `E mail`

X509 Certificate



Liste des extensions

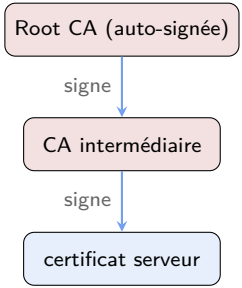
Informations ou contraintes d'utilisation

- **Key Usage**
- **Subject Key Identifier** / **Authority Key Identifier**
- **Subject Alternative Name (SAN)** : noms autres que le DN (formes *GeneralName*)
  - **rfc822Name** : adresse email
  - **dNSName** : nom DNS d'une machine
  - **uniformResourceIdentifier** : URL
  - **iPAddress** : adresse IP

En pratique

Le navigateur vérifie notamment que le `dNSName` correspond au **domaine** demandé dans la barre d'adresse (sinon "certificat invalide").

Hiérarchie et chaîne



Certificat racine (Root CA)

- clés publiques **non signées** ou **auto-signées** ; le **sommet de la pyramide de confiance**
- rarement obtenu par la racine : elle émet des **CA intermédiaires** (conservées sous scellés / déconnectées)

CA de confiance

- importées par défaut dans le navigateur
- être importée passe par **payer** / se soumettre aux exigences des navigateurs

Chaîne de certification (chain of trust)

- les **CA intermédiaires** signent les certificats clients et d'autres CA intermédiaires
- le serveur doit alors fournir la **chaîne de certification** au cas où l'intermédiaire n'est pas dans le navigateur

CA connues

DigiCert (Verisign), Comodo, ... et gratuites : Let's Encrypt (fondée par l'EFF et Mozilla), CAcert (non reconnue), StartSSL

Certificate Revocation List (CRL)

Révocation de certificats

- liste des **identifiants** des certificats révoqués / invalidés
  - utile en cas de **compromission** / décommissionnement
  - la CA ajoute le certificat à la liste
  - la liste est **signée** par la CA
- souvent la même clé signe certificats et CRL
- **quand** le navigateur interroge-t-il les CRL ?

Contenu d'une CRL

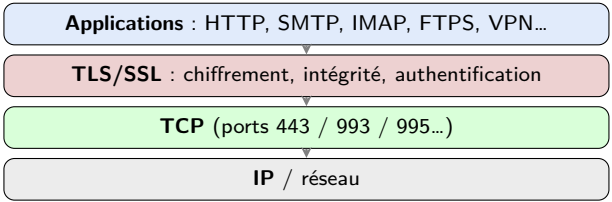
Serial, algorithme de signature, **Issuer**, **Update date** (*thisUpdate*), **Next Update date** (*nextUpdate*), puis liste des certificats révoqués (avec leur motif), extensions.

Motifs de révocation (RFC 5280)

`unspecified` (0), `keyCompromise` (1), `CACompromise` (2), `affiliationChanged` (3), `superseded` (4), `cessationOfOperation` (5), `certificateHold` (6), `removeFromCRL` (8), `privilegeWithdrawn` (9), `AACompromise` (10) (*valeur 7 non utilisée*)

Section 3 : **TLS et X.509**

Place dans le modèle TCP/IP



couche intermédiaire indépendante du service

- couche **intermédiaire**, indépendante du protocole transporté ⇒ réutilisée par HTTPS, IMAPS, SMTPS...(port standard : HTTPS = 443)
- initialement pensé pour HTTP, étendu à d'autres services : SMTP, IMAP, LDAP, VPN, ...

Versions de SSL/TLS

Secure Socket Layer (Netscape)

- **1.0** (1994) : jamais rendue publique
- **2.0** (fév. 1995) : draft SSL 2.0
- **3.0** (nov. 1996) : draft SSL 3.0

Transport Layer Security (IETF)

- **1.0** (jan. 1999, RFC 2246) : TLS 1.0 = SSL 3.1
- **1.1** (avr. 2006, RFC 4346)
- **1.2** (août 2008, RFC 5246)
- **1.3** (août 2018, RFC 8446) : handshake plus rapide (1-RTT), récupération de session et **PFS** par défaut

Recommandations

ANSSI : privilégier TLS 1.3, tolérer TLS 1.2, proscrire SSL et TLS anciens (*downgrade*).

Suite cryptographique

- **échange de clés**
- **authentification** des parties
- **chiffrement** des données applicatives
- protection en **intégrité** (MAC)

```
$ openssl ciphers -v
```

Exemple : TLS_RSA_WITH_AES_128_CBC_SHA256

- **RSA** : authentification du serveur et transport du secret
- **AES_128_CBC** : confidentialité
- **SHA256** : intégrité (MAC)

clienthello

Le client envoie les **suites cryptographiques** qu'il peut mettre en œuvre et ses paramètres (random).

Réponse du serveur

- si aucune suite satisfaisante : **Alert** (fin de connexion)
- sinon :
  - **ServerHello** : suite cryptographique choisie
  - **ServerRandom** : valeur aléatoire (publique)
  - **Certificate** : certificat du serveur
  - **ServerHelloDone** : en attente de la réponse du client

Négociation et fin de handshake

Négociation de la suite cryptographique

- **IIS** : première suite préférée du serveur *et* proposée par le client
- **Apache** : première suite proposée par le client supportée par le serveur → possible **baisse de sécurité** (`SSLHonorCipherOrder` pour sécuriser)

Le client → serveur

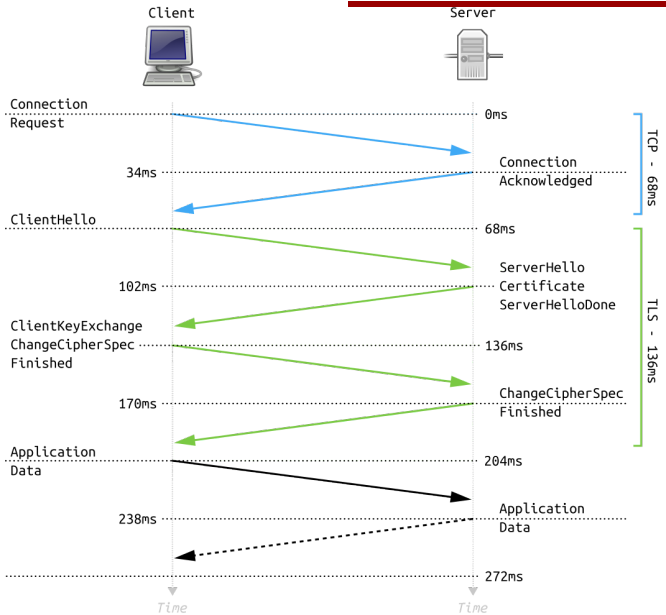
- vérifie la **validité du certificat** ; génère un *pre-master secret* et le chiffre avec la **clé publique** du serveur
- envoie `ClientKeyExchange`, `ChangeCipherSpec`, `Finished` (premier message chiffré)

Le serveur

déchiffre le secret avec sa clé privée puis envoie `ChangeCipherSpec` + `Finished`.

La suite de la communication est chiffrée **symétriquement**.

Handshake TLS 1.2 en image



Construction du handshake TLS 1.2 (AKE)

Protocole complet (noté τ = transcript)

- Client C choisit un nonce N_C et une clé éphémère KE_C
- $C \to S$: N_C, suites cryptographiques, extensions
- S génère N_S, possède (PK_S, SK_S)
- $S \to C$: N_S, Cert_S, suites, extensions avec $\text{Cert}_S = \{S, PK_S\}_{\text{CA}}$ (signé par la CA)

Dérivation des clés

1. le client vérifie Cert_S et calcule le *pre-master secret* pmk (selon le mode RSA/DH/DH Ephemeral (DHE))
2. **master secret** :
$$msk = \text{HMAC}_{pmk}(N_C \| N_S)$$
3. clés de session (chiffrement + MAC par sens) :
$$K_C \| K_S = \text{PRF}_{msk}(0 \| N_C \| N_S)$$

Vérification (τ = transcript)

$C \to S : KE_C \,\|\, \{\mathsf{Fin}_C\}_{K_C}$, où $\mathsf{Fin}_C = \mathsf{HMAC}_{msk}(1\|\tau)$; le serveur vérifie puis renvoie $\{\mathsf{Fin}_S\}_{K_S}$ avec $\mathsf{Fin}_S = \mathsf{HMAC}_{msk}(2\|\tau)$.

Concrètement

le *pre-master secret* pmk (secret de session) devient msk, puis $K_C\|K_S$: les clés **symétriques** de session (chiffrement + MAC) utilisées par le **record layer** pour le volume de données.

Les 3 modes de calcul du pre-master key

Mode RSA (le plus répandu en TLS 1.2)

le client choisit $pmk \in_R \{0,1\}^{384}$; envoie $KE_C = \mathsf{RSA}_{PK_S}(pmk)$; le serveur retrouve pmk avec SK_S. Inconvénient : pas de PFS (si SK_S compromise, les sessions enregistrées sont déchiffrables).

Mode DH (fixe)

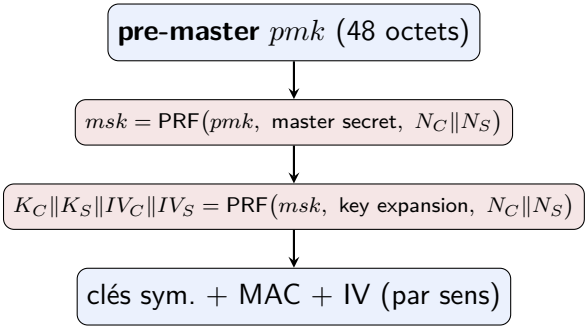
$KE_S = g^{ke_S} \bmod p$ **statique** dans le certificat ; le client choisit ke_C et $pmk = KE_S^{ke_C} \bmod p$.

Mode DHE (éphémère) → PFS

- $KE_S = g^{ke_S} \bmod p$ **frais** à chaque session ; le serveur le **signe** avec SK_S pour authenticité
- client : $KE_C = g^{ke_C}$, $pmk = KE_S^{ke_C} \bmod p$; serveur : $pmk = KE_C^{ke_S} \bmod p$

DHE/ECDHE détruisent les clés éphémères à la fin de la session → la compromission future de SK_S ne déchiffre pas le passé (**PFS**).

De pmk aux clés de travail



- la **PRF** repose sur `HMAC`
- les **nonces** N_C, N_S garantissent la fraîcheur (anti-*rejeu*) : jamais deux sessions identiques
- une **reprise de session** réutilise msk pour éviter un coûteux échange asymétrique

Couche de base de TLS

- toutes les données applicatives passent dans des **records** TLS (max $\sim$ 16 Ko)
- chaque record est chiffré (**symétrique**) et authentifié

MAC-then-Encrypt (TLS $\leq$ 1.2)

$$\text{exp} \quad \longrightarrow \quad \text{Enc}_K\big(\underbrace{m \parallel \text{MAC}_{K_m}(m)}_{\text{type} + \text{seq} + \text{longueur}}\big)$$

- calcule le **MAC**, puis **chiffre** le tout
- vulnérable aux attaques **padding oracle** (ex. **POODLE** sur SSL 3.0/CBC)

AEAD (TLS 1.2+ / TLS 1.3)

$$(c, \tau) = \text{AES-GCM}_K(N, m, \text{hd})$$

- chiffrement et authentification en **un seul passage** (GCM, ChaCha20-Poly1305)
- **TLS 1.3** n'autorise **que** de telles suites $\rightarrow$ élimine les attaques de *padding*.

Authentification mutuelle optionnelle

- dans le handshake, le **client** a authentifié le serveur (certificat serveur) mais le serveur n'a aucune information sur le client → possibilité de **certificat client**

Certificats clients

- le client présente un **certificat** au serveur
- le serveur vérifie qu'il est signé par une **CA de confiance**
- utilisable pour **authentifier l'utilisateur**
- stocké dans un périphérique (*YubiKey*), une carte à puce (*smart card*), ...

Sans PFS (RSA statique)

Si la **clé privée** du serveur est récupérée plus tard : on retrouve le secret de session et on dérive **toutes** les clés de session → communications **passées et futures** déchiffrables.

Avec PFS (Diffie-Hellman éphémère)

- la compromission de la clé privée **n'affecte pas** la confidentialité des communications **passées**
- la clé privée sert uniquement à **signer**
- chaque session établit un secret via **Diffie-Hellman** éphémère

Préférer les suites offrant la **PFS** (ECDHE / DHE) → protection du passé.