Département
Informatique
EPN 05

Louis JOBERT

RSX112, Sécurité des réseaux, M. PIOCH

Cnam Paris, semestre 2, 26 juin 2022


TLS 1.3

TLS 1.3, protocole, nouveautés et différences avec la version 1.2

TABLE DES MATIÈRES

Introduction : SSL/TLS, bref historique du protocole

- Secure sockets layer (couche de sockets sécurisée) et Transport layer security (sécurité de la couche transport) sont des protocoles de sécurisation d'échanges par internet.
- Développé par Netscape, SSL a été développé sous sa première forme en 1994, il a été réellement implémenté en 1995 dans sa seconde version. SSL3.0 est paru en 1996 et fut abandonnée en 2014 suite à la découverte d'une faille de sécurité.
- TLS, dans sa première version en 1999, fut développé par l'IETF (Internet Engineering task force) organisme de standardisation, afin de succéder à SSL, 5 ans et 28 brouillons ont été nécessaires à l'achèvement de la version 1.3.
- L'objectif initial du développement de ces protocoles était d'accroître la sécurisation des paiements en ligne, d'accroître ainsi la confiance dans les transactions et donc de développer le commerce en ligne.

Partie 1 :
Description du protocole TLS 1.3

Qu’est ce que TLS

- TLS est un protocole de sécurisation de communication de la couche application. Il permet de de fournir un canal sécurisé entre deux pairs communicants.
- Il fonctionne suivant un modèle client ↔ serveur.
- Les fonctionnalités de sécurité apportées par TLS :
  - La confidentialité, obtenue par le chiffrement, les données applicatives sont encapsulées (cryptographie symétrique et encapsulation asymétrique). Les données envoyées sont uniquement visibles par les client / serveur.
  - L’intégrité des données applicatives, les données envoyées sur le canal ne peuvent pas être modifiées.
  - L’authentification, le côté serveur est toujours authentifié, l’authentification côté client est optionnelle.
  - La non réutilisation

SSL/TLS dans le modèle OSI

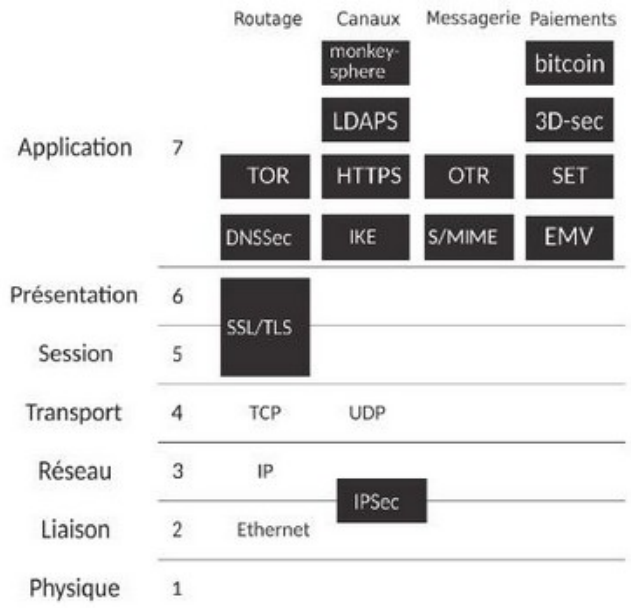
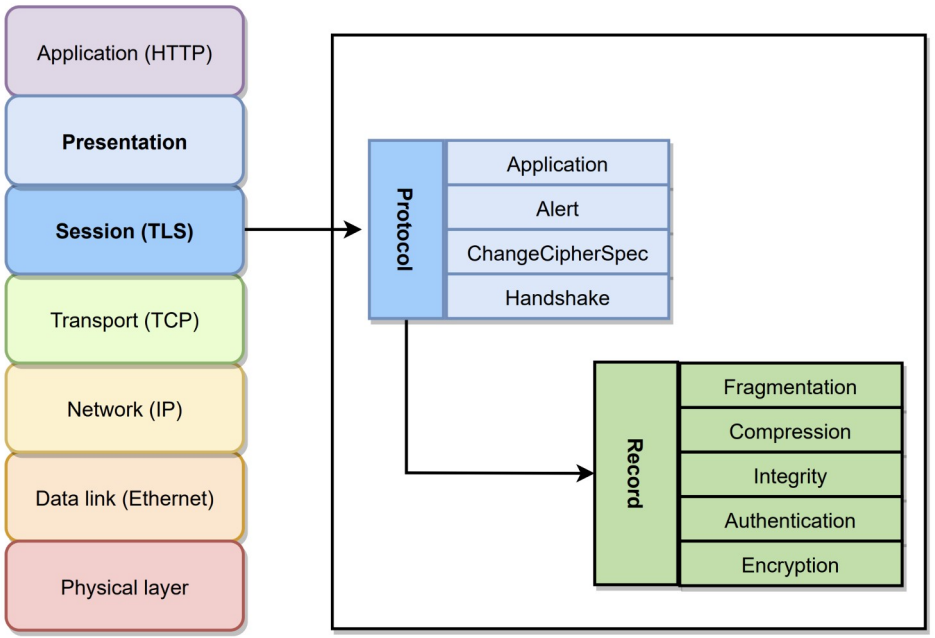


FIGURE 12.1 – Protocoles de sécurité et modèle OSI.

- Le protocole TLS, de niveau 5/6 se situe entre la couche application et la couche transport.
- Le protocole TLS étant implémenté par la couche session, de ce fait pour toute application qui utilise TCP, en existe une qui utilise SSL. Son utilisation la plus connue est HTTPS.
- Une application TLS se voit attribuer un numéro de port par l'IANA. Dans l'exemple précédent HTTPS est associé au port 443, HTTP le 80. Un port supplémentaire est donc défini afin de discriminer le trafic entre la version non sécurisée et celle avec TLS. Concernant d'autres protocoles comme SMTP, POP ou IMAP le même port est utilisé avec et sans TLS, la connexion est dans ces cas initiée en mode non chiffrée. Le tunnel est ensuite mis en place au moyen du mécanisme StartTLS.

Description protocolaire de TLS

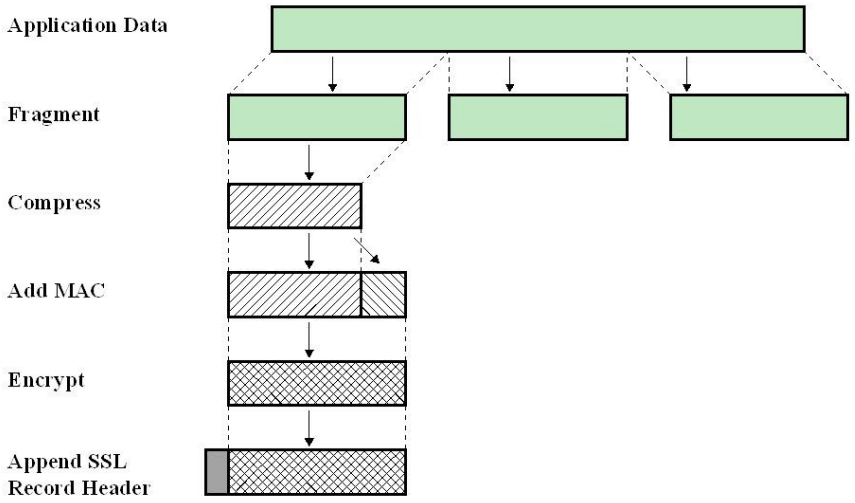


Les messages transmis par l'intermédiaire du protocole TLS sont appelés records. Ils sont généralement encapsulés dans des segments TCP.

Le protocole TLS 1.3 se déroule en plusieurs sous-protocoles distincts :

- Le protocole Handshake, phase de mise en place d'une connexion sécurisée entre le client et le serveur
- Le protocole Alert, il permet soit au client, soit au serveur, d'indiquer la survenue d'une erreur
- Le protocole Record
- Le protocole Application Data « Données de niveau application »

Sequencement du protocole TLS record



Sous-jacent aux autre protocoles, ce protocole est utilisé pour encapsuler et transporter les données des autres protocoles.
Les paramètres qu'il utilise sont négocié par le client et le serveur dans le Handshake, celui-ci est encapsulé par le Record dès les premiers échanges malgré le fait que les paramètres ne soit pas encore connus. Ses paramètres de sécurité sont :

- Le statut de l'entité (client ou serveur)
- Les algorithmes cryptographiques et tailles de clefs à utiliser, chiffrement et intégrité MAC des autres protocoles
- Le type de chiffrement : AEAD (authenticated encryption with additional data) les données sont simultanément chiffrées et protégées en intégrité par la fonction cryptographique. Les chiffrements en continu (stream cipher), et par bloc, n'étant plus utilisés dans TLS1.3 au détriment de l'AEAD.
- L'algorithme de compression
- Le master secret
- Les valeurs aléatoires clientHello.random et serverHello.random, de 32 octets (256bits), permettant de prévenir les attaques par rejeux

Les données divisées en fragments (TLSPlaintext.fragment) ont une taille maximale de 2^14 bytes, soit 16000 octets.

Les trois phases du Handshake

Client *C* Serveur *S*

…… Échange de clefs ……..

- Première phase d'échange de clef entre le client et le serveur. Négociation d'algorithmes cryptographiques et échange de clé non authentifié : le client propose au travers du ClientHello des suites cryptographiques, des groupes sur lesquels s'effectuera un échange de clef.

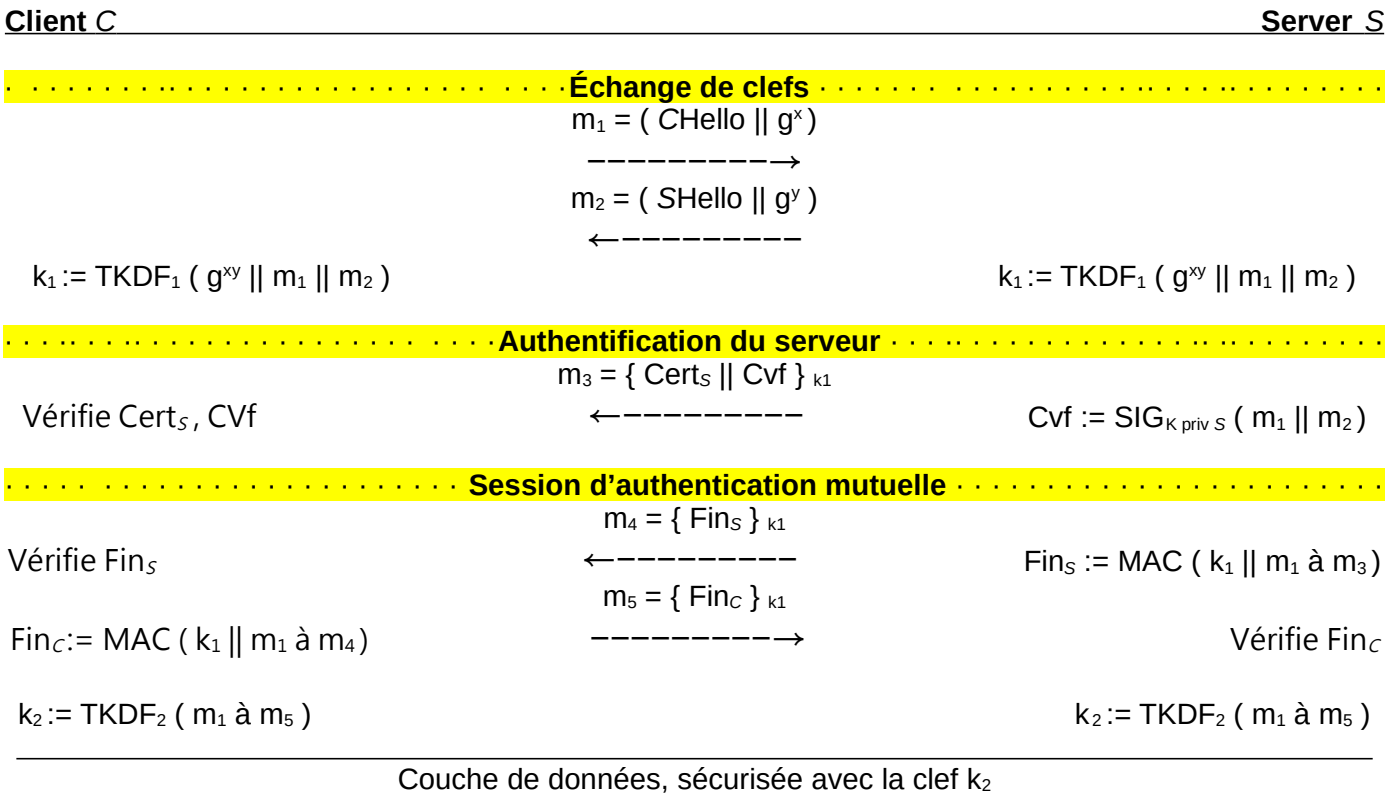
…… Authentification du serveur ……

- Seconde phase d'authentification du serveur au travers de son certificat.

…… Authentification mutuelle ……

- Troisième phase d'authentification entre le client et le serveur, suite à l'échange et la vérification mutuelle de nonces (contenus dans les premiers messages « Hello »).

Le protocole TLS1.3 détaillé



Détail du protocole Alert

```
enum { warning(1), fatal(2), (255) } AlertLevel;

enum {
    close_notify(0),
    unexpected_message(10),
    bad_record_mac(20),
    record_overflow(22),
    handshake_failure(40),
    bad_certificate(42),
    unsupported_certificate(43),
    certificate_revoked(44),
    certificate_expired(45),
    certificate_unknown(46),
    illegal_parameter(47),
    unknown_ca(48),
    access_denied(49),
    decode_error(50),
    decrypt_error(51),
    protocol_version(70),
    insufficient_security(71),
    internal_error(80),
    inappropriate_fallback(86),
    user_canceled(90),
    missing_extension(109),
    unsupported_extension(110),
    unrecognized_name(112),
    bad_certificate_status_response(113),
    unknown_psk_identity(115),
    certificate_required(116),
    no_application_protocol(120),
    (255)
} AlertDescription;

struct {
    AlertLevel level;
    AlertDescription description;
} Alert;
```

La gestion des erreurs dans TLS s'effectue par ce protocole. Lorsqu'une erreur est détectée, la partie détectrice envoie un message à son homologue.
Comme pour les autres messages, les messages d'alerte sont protégés et chiffrés par le protocole Record, ainsi les informations incluses ne fuient pas.

Les types d'erreurs sont catégorisées :

- Suivant le niveau de gravité «AlertLevel» : soit «Warning» soit «Fatal». Une erreur fatale induit la clôture immédiate de la connexion entre les deux parties, et empêchent les renégociations futures à l'aide de l'ID de session en cours.
- Suivant la description de l'alerte « AlertDescription » : soit «Closure Alerts» soit «Error Alerts».

Concernant les «Closure Alerts», qui sont 'user_canceled' et 'close_notify', il est indispensable que client et serveur se donnent l'information d'une clôture de connexion, ceci afin d'éviter une attaque par troncation. Celles-ci peuvent bloquer les demandes de déconnexion afin que l'utilisateur reste connecté à un service web sans le savoir.

Toutes les «Error Alerts» DOIVENT être envoyées avec le niveau de gravité AlertLevel=fatal, quel que soit le niveau d'alerte dans le message.

TLS et le certificat numérique X.509

Après le Client Hello, l'authentification du serveur par le client se fait généralement par l'intermédiaire d'un certificat X.509, celui-ci émis par une autorité de certification. L'authentification du client par le serveur est également possible sans être obligatoire. Lors d'une ouverture d'une connexion TLS vers un site possédant un certificat certifié par une autorité connue, le chemin de certification est validé. Dans le cas inverse, le navigateur propose d'examiner le certificat (certificat généré et auto-signé par un particulier ou autorité de certification pas encore connue ou volontairement retirée de liste des autorités acceptées).

L'ANSSI émet quelques recommandations en ce qui concerne les attributs des certificats X.509 utilisés lors du handshake.

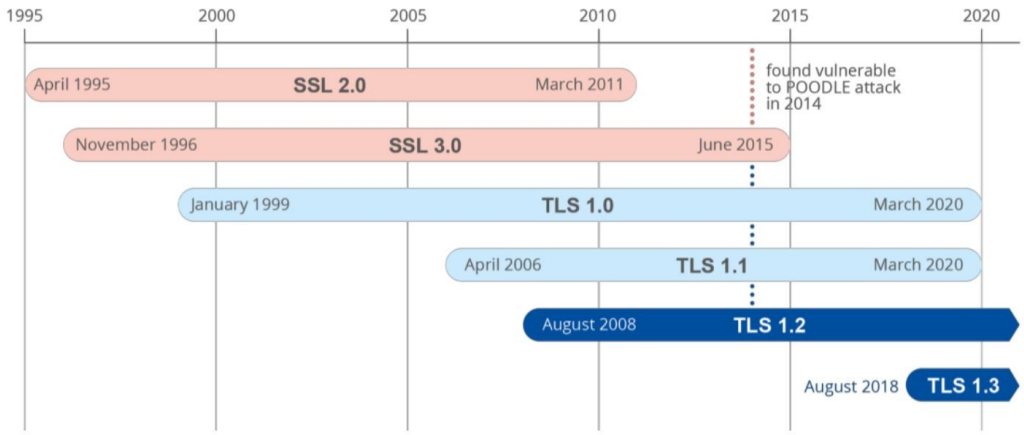
1) Présenter un certificat signé avec SHA-2 :
   La fonction de hachage utilisée pour la signature du certificat doit faire partie de la famille SHA-2.

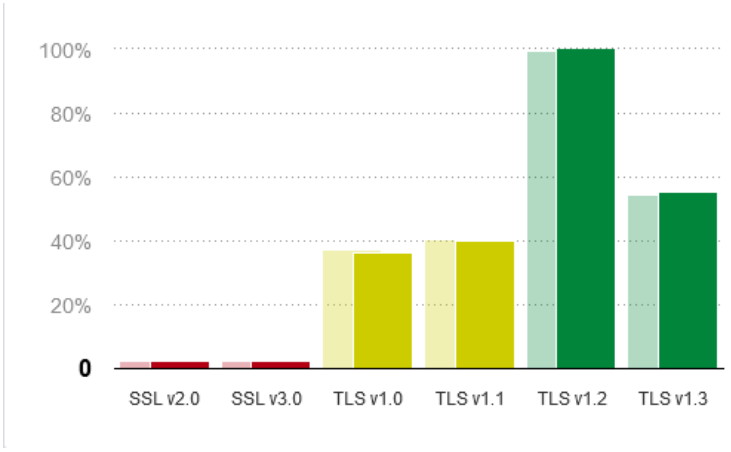
2) Présenter un certificat valide pendant 3 ans ou moins :
   La période de validité d'un certificat d'authentification TLS (serveur ou client) ne doit pas excéder 3 ans..

3) Utiliser des clés de taille suffisante : par exemple pour une protection des communications jusqu'en 2030, les clefs ECDSA doivent avoir une taille minimale de 256 bits.
   Depuis TLS1.3, l'échange de la clef de session doit obligatoirement se faire via Diffie-Hellman avec courbes elliptiques (ECDHE) : le RSA ne peut plus être utilisé pour cela. Nous verrons cela dans la seconde partie.

Partie 2 :
Les versions de TLS et leurs évolutions

Comparatifs des versions SSL/TLS





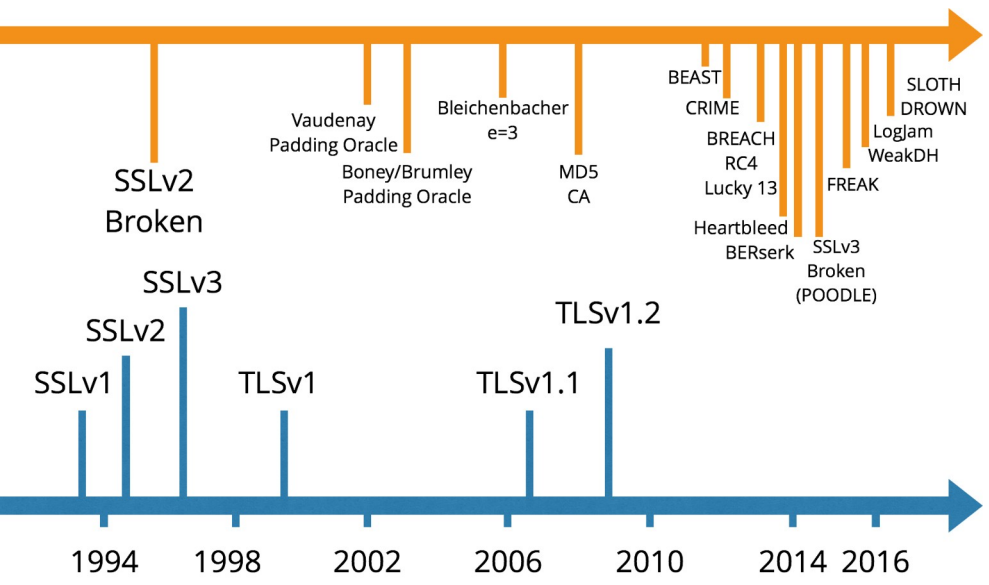
Ci-dessus, les différentes versions de SSL/TLS à travers le temps.

Ci contre, le pourcentages des sites (parmi les plus populaires au monde) supportant les protocoles au 14/06/22, selon SSLLABS. Cela montre qu'une trop grande proportion de sites web acceptent encore des versions obsolètes de TLS, voire même de SSL.

Website protocol support (May 2022)

Protocol version	Website support[71]	Security[71][72]
SSL 2.0	0.3%	Insecure
SSL 3.0	2.5%	Insecure[73]
TLS 1.0	37.1%	Deprecated[8][9][10]
TLS 1.1	40.6%	Deprecated[8][9][10]
TLS 1.2	99.7%	Depends on cipher[n 1] and client mitigations[n 2]
TLS 1.3	54.2%	Secure

Les attaques menées contre TLS



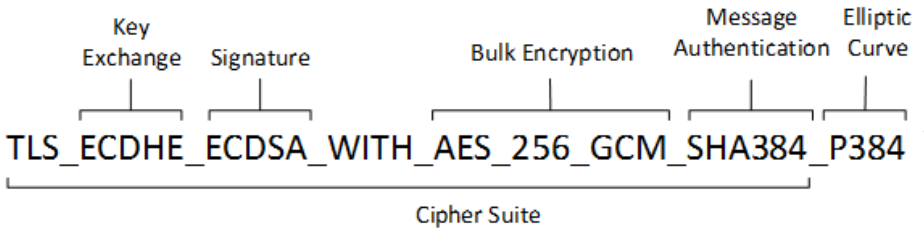
Diverses attaques ont été trouvées sur les protocoles SSL/TLS, les plus connues étant :
Heartbleed en 2012 faille exploitant une mauvaise implémentation et permettait de voler les clés privées des serveurs sécurisés, d'écouter des conversations ou même d'usurper des services.
CRIME (***C***ompression ***R***atio ***I***nfo-leak ***M***ade ***E***asy) en 2012 permettant de récupérer des cookies d'authentification afin de détourner des sessions Web authentifiées.
POODLE (***P***adding ***O***racle ***O***n ***D***owngraded ***L***egacy ***E***ncryption) en 2014 qui exploite le downgrade et le bourrage.
DROWN (***D***ecrypting ***R***SA with ***O***bsolete and ***W***eakened e***N***cryption) en 2016.
Ces attaques exploitent généralement des erreurs d'implémentation, de conception du protocole, de programmation, également des failles sur les primitives cryptographiques.
Cela a amené à l'abandon des versions antérieures à la 1.2 et au développement de la 1.3.
La 1.3 a notamment adopté la suppression d'un algorithme de compression de données sans perte (la Lempel-Ziv77) datant de 1977, celui-ci présent dans la 1.2, pour mettre fin aux attaques par canaux cachés de CRIME.

Les suites cryptographiques dans TLS1.3

Concernant les failles sur les primitives cryptographiques, TLS 1.3 a abandonné le support de certains algorithmes cryptographiques obsolètes et aux nombreuses failles connues, comme par exemple MD5, DSA, SHA-224, DES etc…
Pour en ajouter des nouvelles comme CHACHA20-Poly1305 (stream cipher+MAC), ecdsa_secp256r1_sha256 etc...
La liste des algorithmes de chiffrement pris en charge sont tous des algorithmes de chiffrement authentifié avec données associées (AEAD).

Les suites cryptographiques ont le format suivant :
pour TLS 1.2 : TLS_KE_CIPHER_HASH il diffère pour la version TLS 1.3 : TLS_CIPHER_HASH
KE (Key Exchange) est l'algorithme d'échange de secret, CIPHER l'algorithme de chiffrement symétrique, et HASH une fonction de hachage. La fonction HMAC est dérivée de la fonction de hachage.
L'ANSSI recommande les suites cryptographiques dans le tableau ci pour TLS1.3 dans les cas généraux.

TLS_AES_256_GCM_SHA384 =>(indique que pour TLS on utilise l'algorithme de chiffrement symétrique AES à 256 bits en mode GCM (**G**alois **C**ounter **M**ode étant un mode d'opération de chiffrement par bloc de 128 bits utilisé en cryptographie symétrique) et la fonction de hachage SHA384.



Suite cryptographique
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_AES_128_CCM_SHA256
TLS_CHACHA20_POLY1305_SHA256

Échange de clefs et algorithmes dans TLS1.3

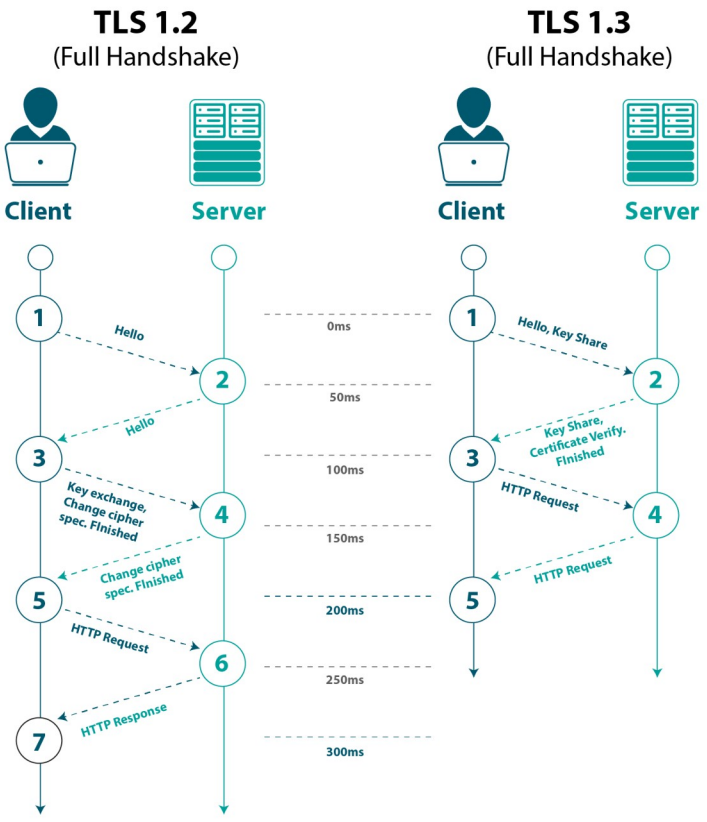
- Seuls les échanges de clefs reposant sur un échange Diffie-Hellman éphémère (ECDHE ou à défaut DHE) sont supportés dans la version 1.3, excluant en pratique le chiffrement RSA, ce qui permet d'assurer ainsi la confidentialité future.
  L'ANSSI recommande l'échange de clef avec l'algorithme ECDHE, en utilisant les groupes 256bits, mais tolère le DHE en utilisant les groupes 2048bits, 3072bits si les données doivent être protégées au-delà de 2030.

- Concernant l'échange de clef, TLS1.3 permet au client de négocier les groupes DHE et les courbes elliptiques ECDHE, la clef ne peut donc plus être définie de manière arbitraire par le serveur, mais doivent faire partie d'une liste de groupes prédéfinis par le client.

- Les suites de chiffrement RSA statique et Diffie-Hellman cipher suites ont été supprimées ; les protocoles d'échanges de clefs publiques doivent intégrer la confidentialité future.

- Les algorithmes de courbe elliptique sont maintenant dans la spécification de base fournies par l'IETF, et les nouveaux algorithmes de signature numérique (DSA **D**igital **S**ignature **A**lgorithm) tels que EdDSA (**Ed**wards-curve **D**igital **S**ignature **A**lgorithm) et ECDSA (**E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm) sont autorisés et inclus.

Différences protocolaires entre les deux versions

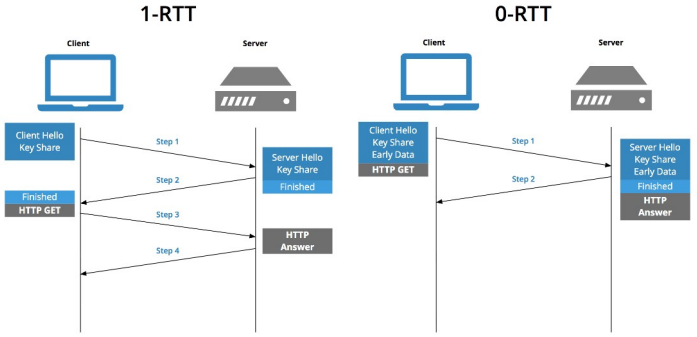


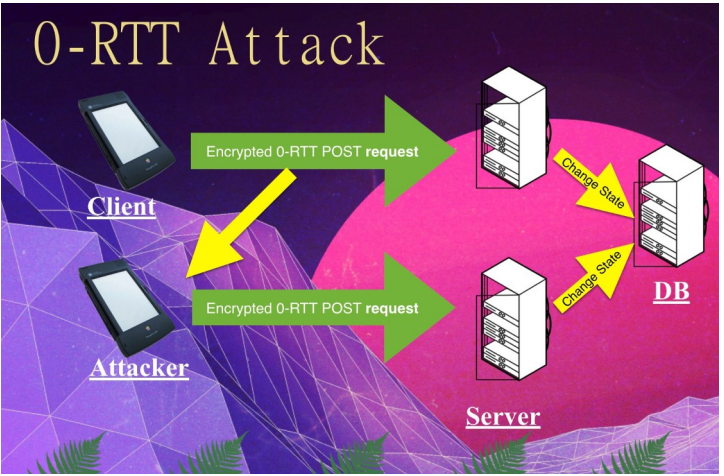
Modification majeure entre les deux versions, la négociation de clefs.
Le Protocole Handshake est plus rapide et comporte moins d'étapes.
Trois échanges de messages pour la version 1.3 au lieu de cinq pour la 1.2.
Le temps de négociation client / serveur est ainsi réduit d'un tiers, environ 200 millisecondes au lieu de 300ms, pour des processeurs standards.

Le protocole TLS1.3 Handshake a été restructuré, afin de simplifier les derniers échanges du protocole Handshake le protocole ChangeCipherSpec a disparu. Ce protocole, consistait en un message chiffré et compressé, signalait les transitions dans les stratégies de chiffrement. Il permettait d'indiquer au participant distant qu'une nouvelle clef allait être utilisée pour le reste de la session. Il reste tout de de même, et uniquement, accessible à des fins de compatibilité (Middlebox Compatibility Mode), notamment concernant les problèmes liés aux middleboxes. Pour que les communications TLS1.3 ne soient pas trop souvent interrompues par des middleboxes, un mode de compatibilité a été prévu pour que TLS1.3 ressemble à TLS 1.2 et génère des messages ChangeCipherSpec artificiels.

La réduction du nombre d'échanges nécessaires est donc due à la suppression de certains messages superflus présents dans les versions précédentes. Les messages ServerHelloDone ont également été supprimés, ainsi que les messages ClientKeyExchange et Server-KeyExchange qui sont utilisés pour échanger les valeurs publiques Diffie–Hellman.

Les modes 1-RTT et 0-RTT (Round Trip Time)





Une nouvelle fonctionnalité apparaît dans la 1.3 : le 0-RTT (zéro aller-retour). L'objectif de L'IETF était d'améliorer les performances du protocole, dans le full-handshake qui est le mode par défaut, la phase de négociation s'effectue en un aller-retour entre le client et le serveur (1-RTT, one round trip time). Ainsi, lors de la toute première connexion d'un client vers le serveur, il y a négociation complète.

0-RTT permet de reprendre une connexion récemment utilisée sans renégocier le cryptage, en proposant donc des connexions contenant des données chiffrées dès le premier paquet.
Cela consiste à transmettre des données applicatives chiffrée dès le premier échange. Afin d'utiliser ce mode les client / serveur doivent posséder une PSK établie lors d'un échange précédent.

Les difficultés engendrées par ce mode 0-RTT, sont que les clefs de chiffrement 0-RTT ne fournissent pas la propriété de confidentialité future, et qu'il est impossible d'avoir une protection anti-rejeu et une distribution fiable de ces données.

L'ANSSI recommande de ne pas envoyer de données via ce protocole, et qu'un serveur n'accepte pas de réception de données 0-RTT, les données ne disposant pas de protection contre les attaques par rejeu, un attaquant peut donc rejouer les messages interceptés. Cet aller-retour économisé s'effectue au détriment de certaines propriétés de sécurité.

Le schéma de gauche montre une attaque par rejeu, en dupliquant simplement un vol de données 0-RTT.

La reprise de session ‘session resumption’

- La session resumption consiste à démarrer une nouvelle session en évitant la renégociation des clefs issue d’une session précédente, cela permet donc d’éviter de repasser par toutes les étapes du Handshake. Elle permet donc d’obtenir rapidement de nouvelles clefs en utilisant celles échangées lors de cette session précédente.

- Dans la version TLS1.3, un ticket est dérivé de la session passée, si ce ticket reçu par le serveur est valide, la phase d’authentification est omise, deux modes sont alors possible, le PSK (**P**re **S**hared **K**ey), ou celui plus sécurisé PSK-DHE (**D**iffie **H**ellman **E**phemeral).

- Ce choix de session resumption se fait par le client lors du premier message envoyé, ou il choisit soit cette session resumption soit le mode normal du protocole Handshake.

- La version TLS1.2 proposait deux méthodes de reprise de session (session resumption). La première reposait sur un ID de session contenu dans le ClientHello, associé à des paramètres et des secrets mis en cache. La seconde s’appuyant sur des tickets de session et d’extensions correspondantes. Ces méthodes ont disparues de la nouvelle version 1.3.

Ressources bibliographiques

- Architectures de sécurité pour internet, Jean-Guillaume Dumas, édition Dunod
- Arithmétique et cryptologie, Gilles Bailly-Maitre, édition Ellipses
- Réseaux et transmissions, Stéphane Lohier, édition Dunod
- Introduction à la cybersécurité, Camille De Sagazan, édition Ellipses
- Initiation à la cryptographie, Gilles Dubertret, édition Vuibert
- Cybersécurité, Solange Ghernaouti, édition Dunod

Ressources en ligne

- https://www.ssi.gouv.fr/uploads/2017/07/anssi-guide-recommandations_de_securite_relatives_a_tls-v1.2.pdf
- https://fr.wikipedia.org/wiki/Transport_Layer_Security
- https://www.rfc-editor.org/info/rfc8446
- https://dev.to/techschoolguru/a-complete-overview-of-ssl-tls-and-its-cryptographic-system-36pd
- https://eprint.iacr.org/2019/749.pdf
- https://www.ssi.gouv.fr/uploads/2015/06/SSTIC2015-Article-ssltls_soa_reloaded-levillain_cObDbqp.pdf
- https://ldapwiki.com/wiki/TLS%201.3
- https://connect.ed-diamond.com/MISC/misc-105/rfc-8446-tls-1.3-que-faut-il-attendre-de-cette-nouvelle-version

Lexique

CHello : Le message « Client Hello » se compose de la version du protocole, d'un nonce généré par le client, ainsi que d'une liste de primitives cryptographiques, et des extensions client.
SHello : le message « Server Hello » inclut un nonce généré par le serveur, ainsi que de la sélection par le serveur de la version, des extensions, et des primitives cryptographiques supportées les plus fortes (parmi les alternatives énoncées dans CHello).
CertS : Le certificat du serveur CertS est modélisé comme une simple clé publique, elle n'est attribuée qu'à une seule entité légitime détenant le clé privée correspondante.
CVf : Le serveur émet le message Certificate Verify pour s'authentifier auprès du client en tant que propriétaire de la clé dans CertS . Le CVf est une signature sur le hachage des messages Handshake, en incluant CertS.
FinS : Le message « Terminé » du serveur est FinS, c'est un MAC lié avec la clef k1.
FinC : Le message FinC (Finished message Client) est un MAC lié avec la clef k1.
STicket : à la fin du Handshake, le serveur peut envoyer une session à un client ticket de reprise STicket suivi d'un nonce NT . Le STicket encapsule rms et un nonce NT de manière cryptée et authentifiée. Ces valeurs sera utilisé pour calculer psk. Le NT doit être transmis au client comme bien. Lors de la reprise d'une session, un client envoie le STicket après CHello et KEC.
k1 : première clef dérivée.
k2 : clef finale.
MAC : Message authentification code. Le code d'authentification de message est un code accompagnant des données dans le but d'assurer leur intégrité, en permettant de vérifier qu'elles n'ont pas subies de modifications après une transmission.
TKDF : Tree key derivation function : Arbre de dérivation de clefs. Une fonction de dérivation de clefs est une fonction qui dérive une ou plusieurs clés secrètes d'une valeur secrète comme un mot de passe ou une phrase secrète en utilisant une fonction pseudo-aléatoire. Les fonctions de dérivation de clé peuvent être utilisées pour renforcer des clés en les étirant.
M1, m2 ,... : messages
|| : concatenation
g : générateur d'un groupe **G**, un groupe étant un ensemble **G.**
$\mathbf{g^x}$ **: x** est la clef privée

TLS 1.3