Introduction à HTTPS

Les objectifs de la cryptographie

La cryptographie est un domaine s'attachant à **protéger des messages**.

Elle a trois objectifs principaux : **la confidentialité, l'authenticité et l'intégrité des messages**.

La confidentialité s'obtient par le chiffrement des messages, permettant de rendre un texte clair totalement incompréhensible pour celui qui ne peut déchiffrer le message à l'aide d'une clé (qui peut être une clé privée ou un secret comme nous le verrons).

L'authenticité permet de s'assurer qu'un message a bien été envoyé par **un émetteur particulier**.

L'intégrité permet de s'assurer qu'un message n'a pas été modifié par un tiers. Il repose sur l'utilisation d'une **signature**.

Les algorithmes et mécanismes en cryptographie permettent souvent de s'assurer d'un ou deux objectifs précités. Il faut donc utiliser souvent une combinaison de procédés et / ou d'algorithmes lorsque nous voulons nous assurer des trois objectifs.

Par exemple, l'utilisation de **code d'authentification de message** (`MAC` en anglais), permet de s'assurer de l'intégrité et de l'authentification d'un message mais ne permet pas de rendre un message confidentiel.

Nous avons utiliser des `HMAC` exactement pour cette raison pour la signature de l'identifiant de sessions.

Les cryptographies symétrique et asymétrique

La cryptographie asymétrique, également appelée cryptographie à clé publique, est une méthode de cryptographie utilisant deux clés : une clé publique et une clé privée.

La cryptographie symétrique est une cryptographie utilisant une donnée secrète, souvent appelée secret, partagée entre celui qui envoie et celui qui reçoit un message.

La cryptographie asymétrique

La cryptographie asymétrique permet de réaliser les trois objectifs de la cryptographie à savoir le **chiffrement** (confidentialité) et la **signature** (intégrité et authenticité) des messages.

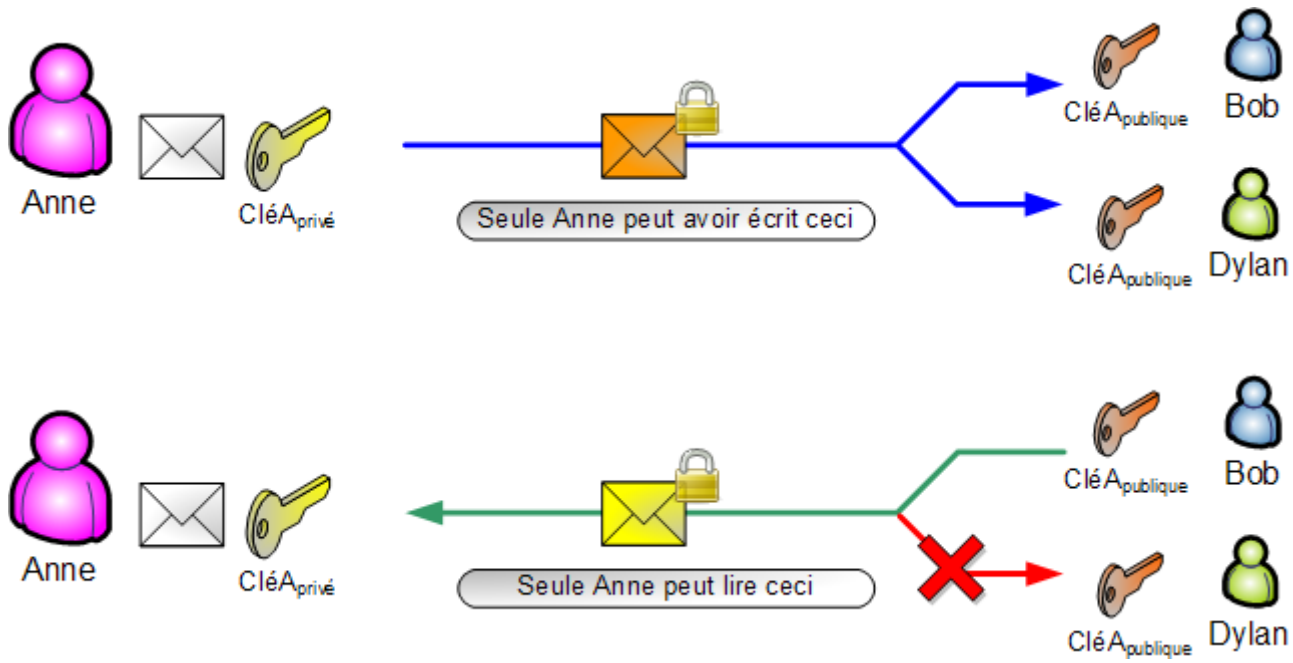
Cela ne veut pas dire que tous les algorithmes de cryptographie asymétrique peuvent réaliser les trois objectifs.

Comme dit précédemment, la cryptographie asymétrique utilise des paires de clés : une publique et l'autre privée.

La **clé publique** est comme son nom l'indique totalement publique, elle peut être propagée à tous. Elle peut soit chiffrer un message, soit vérifier la signature d'un message.

La **clé privée** doit être totalement privée et maintenue secrète. Elle peut soit déchiffrer un message, soit créer une signature d'un message.

Prenons le schéma suivant :



Anne possède une clé privée et une clé publique et souhaite envoyer des messages à Bob et Dylan.

Dans le premier cas, Anne, Bob et Dylan veulent s'assurer de **l'authenticité et de l'intégrité du message**.

Autrement dit, ils veulent s'assurer qu'un message envoyé par Anne ne peut pas être modifié par un tiers, ou envoyé par un tiers se faisant passer pour Anne, sans que Bob et Dylan le sachent.

Pour ce faire, Anne a envoyé sa clé publique à Bob et Dylan avant d'envoyer ses messages. Sa clé publique peut être interceptée par un tiers lors de l'envoi, elle est publique donc cela n'importe pas.

Ensuite Anne va signer ses messages avec sa clé privée. Bob et Dylan pourront s'assurer grâce à la clé publique d'Anne que les messages n'ont pas été modifiés et qu'ils proviennent bien d'Anne.

En effet, **seul le détenteur de la clé privée peut signer un message**. Un **détenteur de la clé publique ne peut que vérifier la signature**. Si un seul caractère du message est modifié par un tiers, la signature ne correspondra plus au message et Bob et Dylan le sauront. Si un tiers

essaye d'envoyer un message avec une fausse signature, Bob et Dylan le sauront également en vérifiant que la signature correspond au message à l'aide de la clé publique.

Dans le second cas, Anne, Bob et Dylan souhaitent s'assurer que les messages envoyés par Bob et Dylan à Anne ne puissent être lus que par Anne.

Pour ce faire, Anne a également envoyé sa clé publique à Bob et à Dylan.

Bob ou Dylan peuvent envoyer des messages qu'ils encryptent avec la clé publique d'Anne.

Mais seule la clé privée d'Anne pourra déchiffrer les messages.

Un tiers interceptant les messages ne pourra pas les déchiffrer. Même Dylan ne pourra pas lire le message chiffré par Bob avec la clé publique d'Anne car il n'a pas la clé privée d'Anne.

Articulation entre cryptographie asymétrique et cryptographie symétrique

Souvent, comme nous le verrons avec `HTTPS`, il est souhaitable d'utiliser la cryptographie asymétrique pour partager un secret entre deux entités de manière sécurité, puis d'utiliser ce secret pour utiliser la cryptographie symétrique.

Pourquoi utiliser la cryptographie symétrique ? Car elle est beaucoup plus rapide que la cryptographie asymétrique, et il est donc plus performant d'envoyer les messages de cette manière après qu'un secret ait été partagé.

Pourquoi ne pas utiliser uniquement la cryptographie symétrique ? Car il est nécessaire d'envoyer le secret de manière sécurisée.

Nous allons détailler le fonctionnement à haut niveau d'un échange de clés en cryptographie asymétrique puis d'un basculement vers la cryptographie symétrique.

Alice et Bob souhaitent s'envoyer une grande quantité de messages, dans les deux sens, et de manière parfaitement sécurisée.

Etape 1 - Alice et Bob génèrent chacun de leur côté leur propre paire de clés publique et privée.

Etape 2 - Alice et Bob s'échangent leur clé publique. Bob a maintenant la clé publique d'Alice et Alice celle de Bob.

Etape 3 - Bob crée une signature d'un message qui contient un secret avec sa clé privée.

Etape 4 - Bob utilise ensuite la clé publique d'Alice pour chiffrer le message et la signature. Il envoie le message et la signature totalement chiffrés à Alice.

Etape 5 - Alice reçoit le message chiffré et le décrypte avec sa clé privée.

Etape 6 - Alice vérifie l'intégrité et l'authenticité du message, grâce à la signature qui a été déchiffrée, en utilisant la clé publique de Bob.

Alice est certaine de **la confidentialité, de l'intégrité et de l'authenticité du message qu'elle a reçu**. Elle peut donc utiliser le secret qu'elle a reçu dans le message en toute tranquillité.

Etape 7 - Alice chiffre un nouveau message en utilisant le secret, nous avons basculé en cryptographie symétrique.

Etape 8 - Bob déchiffre le message en utilisant le secret, maintenant partagé, et renvoie un nouveau message en le chiffrant avec le secret.

Les échanges sont maintenant totalement sécurisés et sont rapides car ils utilisent seulement un chiffrement symétrique.

Qu'est-ce qu'`HTTPS` ?

`HTTP` signfie `HyperText Transfer Protocol Secure`.

En fait il s'agit du protocole `HTTP` avec une couche de sécurité supplémentaire.

Cette couche permet d'une part de **vérifier l'identité du serveur Web** et d'autre part de **transmettre les données de manière confidentielles**.

Historiquement, la partie sécurité était assurée par le protocole `SSL` (pour `Secure Sockets Layer`). Ce protocole n'est aujourd'hui plus utilisé car il n'est plus sécurisé.

Son successeur est le protocole `TLS` (pour `Transport Layer Security`). Sa version `1.3` est sortie en 2018 et est plus rapide et plus sécurisée que jamais, comme nous allons le voir.

Aujourd'hui, 85% du trafic `Internet` est encrypté en `SSL/TLS`.

90% des sites utilisant `HTTPS` utilisent a minima la version `TLS 1.2`.

11% des sites sont réputés comme non sécurisés (version `SSL` et `TLS` anciennes).

Vérification de l'identité du serveur

Pour que le client puisse vérifier que le serveur est bien celui qu'il prétend être, il faut un mécanisme fondé sur l'utilisation d'un **certificat numérique** appelé `certificat X509`.

Le client initie une requête vers `dyma.fr` et effectue donc une requête `DNS` pour savoir quelle(s) adresse(s) `IP` il doit interroger.

Un attaquant pourrait à ce moment intercepter la requête et renvoyer l'`IP` de son serveur et se faire passer pour lui.

Il faut donc un moyen d'authentifier la réponse du serveur pour être certain qu'il s'agit bien d'un serveur possédant le nom de domaine !

Les autorités de certification

Pour cela, le serveur va envoyer un **certificat** délivré par une **autorité de certification (CA)**.

Dans votre navigateur vous avez une liste d'autorités de certification valides avec leurs clés publiques :

Sur Chrome allez dans `chrome://settings/certificates`.

Puis cliquez dans l'onglet `Autorités`.

Prenez un élément au hasard et cliquez sur les trois petits points en colonne, puis sur `Afficher`.

Ensuite allez dans `Détails`.

Vous trouverez plein d'informations dont une **clé publique**, un **algorithme de la clé publique** (par exemple `RSA PKCS #1`).

Votre navigateur a donc toutes les clés publiques de toutes les autorités de certification reconnues par Chrome. Grâce à ces clés, le navigateur peut donc vérifier l'authenticité d'un certificat.

Rappelez-vous qu'une clé publique permet de vérifier une signature et donc vérifier l'intégrité et l'authenticité d'un message !

Authentification des messages : certification de l'identité d'un serveur

Pour se faire certifier, un serveur envoi un ensemble d'informations dont **le ou les noms de domaines qu'il possède** et sa **clé publique**.

L'autorité de certification va ensuite demander au serveur de prouver qu'il possède le domaine, par exemple en plaçant un fichier avec un `token` dans un répertoire `public` servi par le serveur.

Une fois que le serveur l'a fait, l'autorité de certification vérifie le `token` sur une adresse du domaine.

Une fois que la preuve est faite par ce biais ou par un autre, **l'autorité de certification va délivrer un certificat et le signer avec sa clé privée**.

Le certificat contient toutes les informations fournies par le serveur et notamment **les noms de domaines vérifiés** et sa **clé publique**. Il contient également une signature créée avec la clé

privée de l'autorité de certification. Cela signifie que l'intégrité du certificat est assurée : personne ne pourra le modifier.

Est-ce que quelqu'un attaquant peut créer un certificat ? Non, sauf s' il arrive à hacker une autorité de certification. Un attaquant ne pourra pas signer un certificat avec une clé privée valide. Les navigateurs pourront vérifier la signature du certificat avec la clé publique de l'autorité de certification et s'apercevoir qu'elle n'est pas bonne.

Est-ce que quelqu'un attaquant peut utiliser un certificat valide ? Il peut sans problème obtenir le certificat valide du serveur, sauf qu'il n'aura pas accès à la clé privée du serveur qui correspond à la clé publique du certificat. Il ne pourra pas signer l'envoi du certificat au client avec la clé privée du serveur.

Pour résumer, nous avons un certificat contenant des noms de domaines et une clé publique d'un serveur. Nous avons vu que personne, sauf l'autorité de certification, ne peut créer un certificat reconnu par les navigateurs.

Ensuite, la clé privée correspondant à la clé publique contenue dans le certificat signé par l'autorité de certification n'est connue que du serveur.

Envoi du certificat par le serveur au navigateur du client

Etape 1 - Le navigateur envoie une demande de connexion sécurisée à l' `IP` du serveur après une requête `DNS` .

Etape 2 - Le serveur envoi son certificat, signé par l'autorité de certification et il signe le certificat et la signature avec sa clé privée.

Etape 3 - Le navigateur du client vérifie la signature du certificat en utilisant la clé publique de l'autorité de certification contenue dans son navigateur. Il est maintenant certain que le certificat est valide et n'a pas été modifié. (Il vérifie également que le certificat n'a pas été révoqué à l'aide du protocole `OCSP`).

Etape 4 - Le navigateur du client vérifie, avec la clé publique contenue dans le certificat, la signature du message envoyé par le serveur (et qui contient notamment le certificat et sa signature par l'autorité de certification). Il est maintenant certain que c'est bien le serveur qui détient le certificat qui a envoyé le message et le certificat.

Nous avons vu comment le protocole `HTTPS` permettait de vérifier l'identité du serveur. Il nous faut maintenant voir comment la confidentialité des communications est assurée.

Confidentialité et intégrité des communications

Nous allons maintenant aller plus en détails sur le protocole `TLS` qui va permettre d'assurer que **personne ne peut lire les messages** (confidentialité) et que **personne ne peut modifier les**

messages (intégrité).

Le protocole `TLS` est utilisé pour sécuriser un transport `TCP` mais aussi `UDP` ou `DCCP`.

Il permet donc de sécuriser des protocoles comme `HTTP`, `SMTP`, `FTP`, `XMPP`, `NNTP` mais aussi de construire des `VPN` sécurisés.

Globalement `TLS` est le protocole de sécurité le plus utilisé sur `Internet`.

Pour commencer nous allons voir le `handshake TLS`.

Le `handshake TLS`

Les objectifs du `handshake TLS` sont de trois ordres :

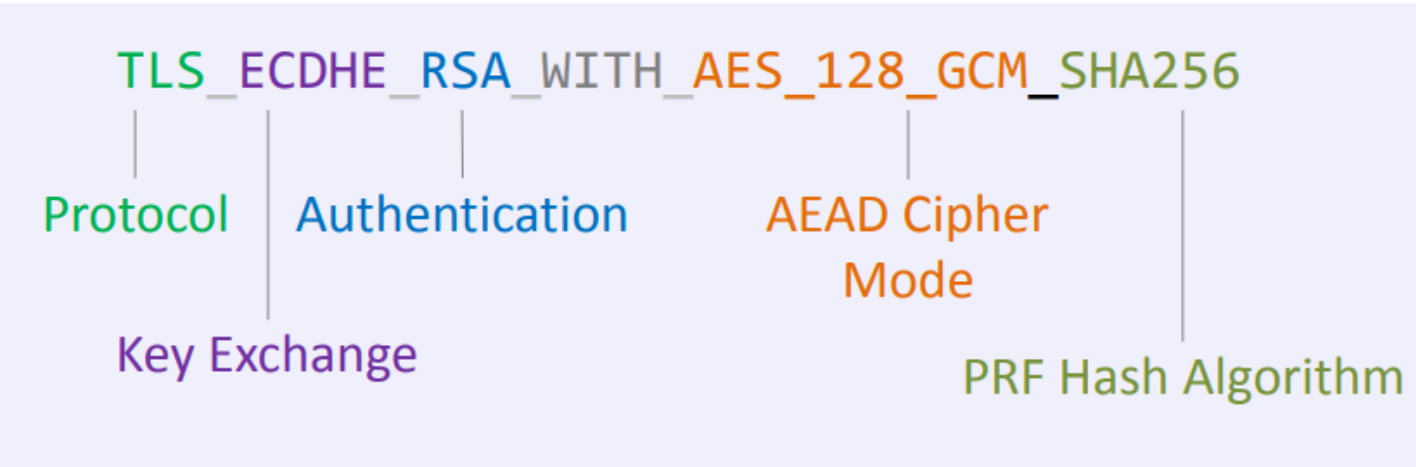
1 - **S'accorder entre le client et le serveur pour utiliser une suite cryptographique commune** (`cipher suite`) constituée par un ensemble d'algorithmes cryptographiques qui vont assurer la sécurité des échanges.

2 - **S'échanger un secret commun** qui sera utilisé pour les échanges chiffrés symétriquement.

3 - **Que le client puisse authentifier le serveur** en utilisant un ou plusieurs certificats. Parfois le serveur doit aussi authentifier le client, mais nous ne le traiterons pas ici.

Etude des suites cryptographiques

Prenons la suite cryptographique la plus utilisée par Google : `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`.



1 - `TLS` est le protocole utilisé pour les échanges sécurisés.

2 - `ECDHE` (`Elliptic curve Diffie-Hellman`) est un algorithme utilisé pour générer et s'échanger un `Pre-Master Secret`. Il permet **la confidentialité persistante** des communications. Cela signifie que même si la clé privée du serveur est compromise, la confidentialité des conversations passées, potentiellement stockées par des attaquants, est assurée.

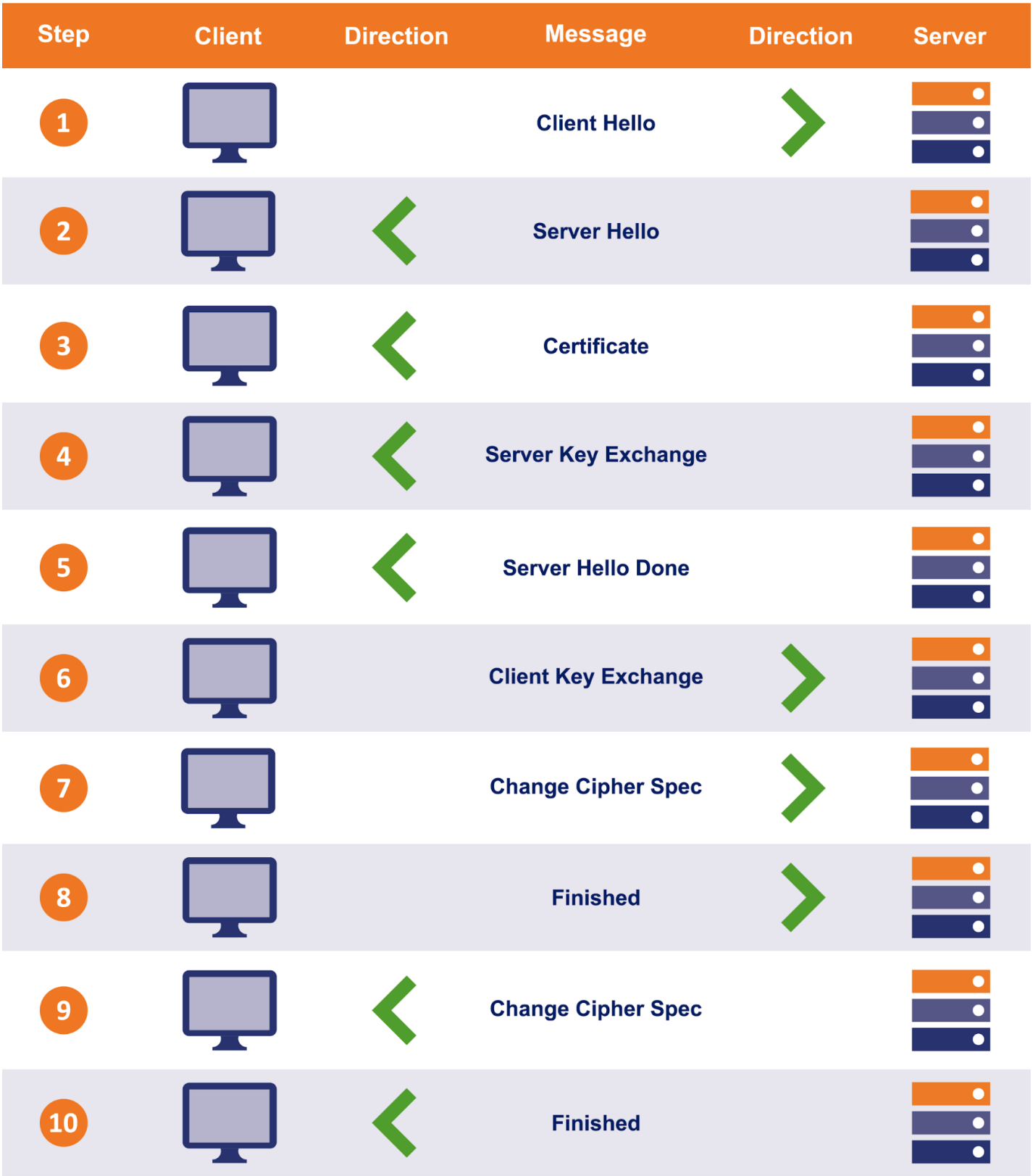
3 `RSA` (`Rivest-Shamir-Adleman`) est l'algorithme utilisé ici pour authentifier les messages lors des échanges de clés.

4 - `AES_128_GCM` (`Advanced Encryption Standard` utilisant `128 bits` et `Galois/Counter Mode`) : ces deux algorithmes permettent d'assurer le chiffrement des messages et l'intégrité des blocs chiffrés. C'est ce qu'on appelle l'`AEAD` (`Authenticated encryption with associated data`). Cela permet de chiffrer les messages mais aussi de s'assurer que le copier coller de blocs chiffrés valides soient détectés. Ce qui est important pour les paquets envoyés sur le réseau.

4 - `SHA-256` (`Secure Hash Algorithm` avec un `hash` de `256 bits`) est utilisée pour la dérivation des secrets à partir du `Pre-Master-Secret`. Ainsi, le secret est unique pour une session `TLS` entre deux entités.

Déroulé d'un `handshake TLS 1.2`

Nous allons commencé par étudier un `handshake TLS 1.2` :



Etape 1 - Le client envoi au serveur un message `ClientHello`. Ce message contient toutes les informations dont le serveur a besoin pour utiliser le protocole `TLS` avec le client (principalement un nombre aléatoire, la liste des suites cryptographiques disponibles pour le client et dernière version `TLS` supportée).

Etape 2 - Le serveur répond avec un `ServerHello` qui contient des informations (notamment un autre nombre aléatoire) et la meilleure suite cryptographique disponible en commun (supportée par le client et le serveur).

Etape 3 - Le serveur envoi son certificat `X509`. Nous avons vu comment le client peut alors être sûr de l'identité du serveur. Le client a donc maintenant la clé publique du serveur.

Etape 4 - Le serveur envoi un message `Server Key Exchange` qui est utilisée par l'algorithme `ECDHE` pour les échanges de clés. Ce message est signé par l'algorithme `RSA` dans notre exemple.

Etape 5 - Le serveur envoi un message de fin.

Etape 6 - Le client envoi un message `Client Key Exchange` qui peut contenir sa clé publique ou le `pre master secret` suivant l'algorithme utilisé. Ce message est chiffré avec la clé publique du serveur et signé avec la clé privée du client.

Etape 7 et 8 - Le client indique au serveur qu'il passe en mode chiffrement symétrique et qu'il a terminé (et que tout s'est bien passé).

Le serveur et le client utilise les deux nombres aléatoires et le `pre master secret` pour générer un `master secret`. Les clés sont dérivés de ce `master secret` en utilisant un algorithme par exemple `SHA256`.

Les clés générées sont ensuite utilisées par les algorithmes de chiffrement symétrique et d'intégrité (par exemple `AES_128_GCM`).

Les 6 clés générées sont les suivantes :

1 - La clé d'écriture client `MAC` : permet au client de signer les données par le client et pour que le serveur puisse vérifier l'intégrité.

2 - La clé d'écriture serveur `MAC` : même chose mais dans l'autre sens.

3 - La clé de chiffrement client : le client utilise cette clé pour chiffrer les données envoyées au serveur.

4 - La clé de chiffrement serveur : même chose mais dans l'autre sens.

5 - La clé d'écriture IV client : cette clé est utilisée par le serveur pour les suites cryptographiques utilisant l'`AEAD`.

6 - La clé d'écriture IV serveur : cette clé est utilisée par le client pour les suites cryptographiques utilisant l'`AEAD`.

Etape 9 - Le client envoi un `Change Cipher Spec` ce qui signifie le passage en chiffrement symétrique.

La version `TLS 1.3`

La nouvelle version du protocole `TLS` est arrivée en 2018, 10 ans après `TLS 1.2` !

Cette version permet une vitesse accrue pour le `handshake TLS` pouvant faire gagner environ 0.5 seconde avant les premières données transférées. Elle y est parvenue en réduisant les échanges nécessaires pour le `handshake`.

Elle permet également une sécurité renforcée : elle n'admet que cinq suites cryptographiques réputées comme étant les plus sécurisées.

Elle utilise que des suites cryptographiques utilisant l'`AEAD` et la confidentialité persistante (`PFS` ou `Persistance Forward Secrecy`) que nous avons vus plus haut.

Elle permet enfin d'empêcher les attaques par renégociation du protocole.

Nous allons d'abord voir le nouveau `handshake` :

Nous ne détaillerons pas une nouvelle fois les principes déjà vue dans le `handshake` précédent comme la dérivation des secrets.

Nous allons simplement détailler brièvement les trois étapes du nouvel `handshake` :

Etape 1 : le client envoi des données générées aléatoirement (nous verrons pourquoi), une liste des suites cryptographiques qu'il supporte, une liste de clés publiques qui peuvent être utilisées par le serveur suivant la suite choisie, et la version des protocoles que le client supporte.

Etape 2 - Le serveur répond avec la suite cryptographique et la version du protocole choisis, son certificat, une clé publique suivant l'algorithme utilisé pour l'échange de clés et des données générées aléatoirement.

Etape 3 - Le client vérifie le certificat, génère les secrets spécifiques à la suite en utilisant la clé publique du serveur, sa clé privée et les données aléatoires. Le serveur fait de même de son côté.

Etape 4 - Basculement en chiffrement symétrique.