Architectures matérielles, systèmes d'exploitation et réseaux: sécurisation des communications:

Soit 2 individus A et B qui cherchent à s'envoyer des messages par l'intermédiaire d'un réseau informatique. A et B désirent qu'une tierce personne (par exemple P) ne soit pas capable de lire les messages si par hasard ces derniers devaient être interceptés par P. Pour ce faire, A va chiffrer le message . Toute personne qui ne possédera pas le moyen de déchiffrer ce message chiffré se verra dans l'impossibilité de comprendre le contenu du message (si P intercepte le message chiffré et qu'il ne possède pas le moyen de déchiffrer ce message, l'interception aura été totalement inutile puisque P sera dans l'incapacité de comprendre le contenu du message). Il existe 2 grands types de chiffrement: le chiffrement symétrique et le chiffrement asymétrique.

1. **Le chiffrement symétrique:**
   Pour chiffrer un message, A va utiliser une suite de caractère que l'on appelle "clé de chiffrement". Dans le cas du chiffrement symétrique, cette clé de chiffrement sera aussi utilisée par B pour déchiffrer le message envoyé par A. Dans ce cas, la clé de chiffrement est identique à la clé de déchiffrement. Concrètement comment cela se passe-t-il?

   Avant d'entrer dans le vif du sujet, il faut savoir que l'idée de chiffrer des messages (de les rendre illisibles pour des personnes non autorisées) ne date pas du début de l'ère de l'informatique. En effet, dès l'antiquité, on cherchait déjà à sécuriser les communications en chiffrant les messages sensibles (pour en savoir plus sur l'histoire du chiffrement, n'hésitez pas à consulter la page Wikipédia consacrée à ce sujet). Nous nous intéresserons ici uniquement aux communications ayant lieu par l'intermédiaire d'un réseau informatique. Comme nous avons déjà eu l'occasion de le voir en première, toute "donnée informatique" peut être vue comme une suite de zéro et de un. Nous chercherons donc à chiffrer une suite de zéro et de un. Soit le message "Hello World!" ce qui nous donnera en binaire:

```
01001000011001010110110001101100011011110010000001010111011011101110010011011000110010000100001
```

   NB: nous avons simplement utilisé le code ASCII de chaque caractère (par exemple, on peut vérifier que le H correspond bien à l'octet 01001000). Pour effectuer la "conversion" texte vers code binaire ASCII ou vis versa, vous pouvez utiliser le site `https://www.rapidtables.com/convert/number/ascii-to-binary.html`

Choisissons maintenant un mot (ou une phrase) qui nous servira de clé de chiffrement, prenons pour exemple le mot "toto". "toto" nous donne en binaire:

```
01110100011011110111010001101111
```

Pour chiffrer le message nous allons effectuer un XOR bit à bit. Pour rappel, vous trouverez la table de vérité du XOR ci-dessous:

E1	E2	S
0	0	0
0	1	1
1	0	1
1	1	0

Comme la clé est plus courte que le message, il faut "reproduire" la clé vers la droite autant de fois que nécessaire (si la taille du message n'est pas un multiple de la taille de la clé, on peut reproduire seulement quelques bits de la clé pour la fin du message):

```
⊕ 01001000011001010110110001101100 01101111001000000101011101101111 01110010011011000110010000100001
  01110100011011110111010001101111 01110100011011110111010001101111 01110100011011110111010001101111
  ──────────────────────────────────────────────────────────────────────────────────────────────────────
  00111100000001010000110000000001100011011010011110010001100000000000001100000001100010000001001110
```

Le signe + dans un cercle symbolise le XOR. Après ce XOR on obtient donc la suite de bits suivante:

```
00111100000001010000110000000001100011011010011110010001100000000000001100000001100010000001001110
```

c'est à dire la chaine de caractères suivante (si on cherche à afficher le message chiffré avec un éditeur de texte): O#N

Maintenant ce message est prêt pour être envoyé à son destinataire B. Si P intercepte le message est cherche à le lire avec un éditeur de texte, il obtiendra la suite de caractère O#N.

B a maintenant reçu le message chiffré, il possède la clé (toto), il va donc pouvoir déchiffrer le message en appliquant un XOR entre le message chiffré et la clé (on applique exactement la même méthode que ci-dessus).

```
⊕ 00111100000001010000110000000001 00011011010011110010001100000000 00000110000000110001000001001110
  01110100011011110111010001101111 01110100011011110111010001101111 01110100011011110111010001101111
  ──────────────────────────────────────────────────────────────────────────────────────────────────────
  01001000011001010110110001101100 01101111001000000101011101101111 01110010011011000110010000100001
```

On trouve le code binaire suivant:

```
01001000011001010110110001101100011011110010000001010111011011110111001001101100011001000010000 1
```

Vous pouvez remarquer que nous avons bien retrouvé le code binaire d'origine. Si vous ne voulez pas vous embêter à vérifier bit par bit, vous pouvez utiliser le site précédent qui vous permettra de repasser du code binaire ASCII au texte.

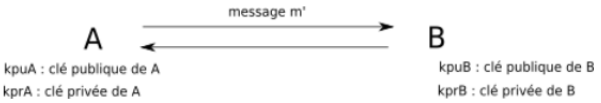
On retrouve bien le message d'origine : Hello World!, B a pu lire le message envoyé par A alors que pour P, malgré le fait qu'il a pu intercepter le message, il n'a pas pu prendre connaissance de son contenu sans la clé.

Exercice: Vous allez jouer le rôle de A. Choisissez une ou un camarade dans la classe qui jouera le rôle de B. Mettez vous d'accord avec B sur une clé de chiffrement/déchiffrement (choisissez un mot qui jouera le rôle de clé, ce mot doit rester secret). Choisissez un message à faire parvenir à B puis procéder au chiffrement de ce message, notez le résultat du chiffrement (en binaire) sur une feuille. Donnez cette feuille à une ou un camarade tiers (qui ne connait pas la clé, ce camarade jouera donc le rôle de P). P devra recopier le message avant de le transmettre à B. P devra essayer de trouver le message envoyer par A à B. B devra déchiffrer le message à l'aide de la clé. Vous pourrez utiliser les sites cités plus haut afin d'assurer le passage texte $\longrightarrow$ code ASCII binaire et code ASCII binaire $\longrightarrow$ texte.

La méthode la plus utilisée en matière de chiffrement symétrique se nomme AES (Advanced Encryption Standard). Cette méthode utilise une technique de chiffrement plus élaborée que ce qui a été vu ci-dessus, mais les grands principes restent identiques. Le gros problème avec le chiffrement symétrique, c'est qu'il est nécessaire pour A et B de se mettre d'accord à l'avance sur la clé qui sera utilisée lors des échanges. Le chiffrement asymétrique permet d'éviter ce problème.

2. **Le chiffrement asymétrique:**
Dans le cas du chiffrement asymétrique A et B n'ont pas besoin de partager une "clé secrète":

   - A possède une "clé privée" que l'on notera kprA et une "clé publique" que l'on notera kpuA. En aucun cas A ne devra divulguer sa clé privée à quiconque, elle devra rester strictement secrète. En revanche sa clé publique pourra être connue de tout le monde sans aucun problème.
   - B possède une "clé privée" que l'on notera kprB et une "clé publique" que l'on notera kpuB. En aucun cas B ne devra divulguer sa clé privée à quiconque, elle devra rester strictement secrète. En revanche sa clé publique pourra être connue de tout le monde sans aucun problème.



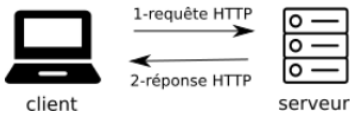
Si A désire envoyer un message m à B, il va utiliser la clé publique de B afin de réaliser le chiffrement (m est chiffré en m'). Le message chiffré (m') va ensuite pouvoir transiter entre A et B. Une fois le message m' en sa possession, B va utiliser sa clé privée afin de pouvoir déchiffrer le message m' et ainsi obtenir le message m. Le processus peut être résumé par le schéma suivant:

Si P intercepte le message m', il sera incapable de déterminer m à partir de m' sans la clé privée de B.

Le chiffrement asymétrique repose sur des problèmes très difficiles à résoudre dans un sens et faciles à résoudre dans l'autre sens. Prenons un exemple : l'algorithme de chiffrement asymétrique RSA (du nom de ses 3 inventeurs : Rivest Shamir et Adleman), est très couramment utilisé, notamment dans tout ce qui touche au commerce électronique. RSA se base sur la factorisation des très grands nombres premiers. Si vous prenez un nombre premier A (par exemple A = 16813007) et un nombre premier B (par exemple B = 258027589), il facile de déterminer C le produit de A par B (ici on a A x B = C avec C = 4338219660050123). En revanche si je vous donne C (ici 4338219660050123) il est très difficile de retrouver A et B. En tous les cas, à ce jour, aucun algorithme n'est capable de retrouver A et B connaissant C dans un temps "raisonnable". Nous avons donc bien ici un problème relativement facile dans un sens (trouver C à partir de A et B) est extrêmement difficile dans l'autre sens (trouver A et B à partir de C). Les détails du fonctionnement de RSA sont relativement complexes (mathématiquement parlant) et ne seront pas abordés ici. Vous devez juste savoir qu'il existe un lien entre une clé publique et la clé privée correspondante, mais qu'il est quasiment impossible de trouver la clé privée de quelqu'un à partir de sa clé publique.

3. **Le protocole HTTPS:**
Nous allons maintenant voir une utilisation concrète de ces chiffrements symétriques et asymétriques: le protocole HTTPS.

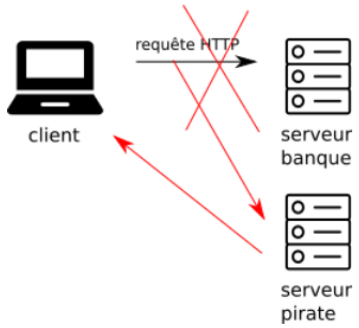
Avant de parler du protocole HTTPS, petit retour sur le protocole HTTP: un client effectue une requête HTTP vers un serveur, le serveur va alors répondre à cette requête (par exemple en envoyant une page HTML au client). Si nécessaire n'hésitez pas à consulter ce qui a été fait en première pour plus de détails (`https://pixees.fr/informatiquelycee/n_site/nsi_prem_http.html`).



échange client-serveur protocole HTTP

Le protocole HTTP pose 2 problèmes en termes de sécurité informatique:

- Un individu qui intercepterait les données transitant entre le client et le serveur pourrait les lire sans aucun problème (ce qui serait problématique notamment avec un site de e-commerce au moment où le client envoie des données bancaires)
- grâce à une technique qui ne sera pas détaillée ici (le DNS spoofing: `https://fr.wikipedia.org/wiki/Empoisonnement_du_cache_DNS`), un serveur "pirate" peut se faire passer pour un site sur lequel vous avez l'habitude de vous rendre en toute confiance : imaginez vous voulez consulter vos comptes bancaires en ligne, vous saisissez l'adresse web de votre banque dans la barre d'adresse de votre navigateur favori, vous arrivez sur la page d'accueil d'un site en tout point identique au site de votre banque, en toute confiance, vous saisissez votre identifiant et votre mot de passe. C'est terminé un "pirate" va pouvoir récupérer votre identifiant et votre mot de passe ! Pourquoi ? Vous avez saisi l'adresse web de votre banque comme d'habitude ! Oui, sauf que grâce à une attaque de type "DNS spoofing" vous avez été redirigé vers un site pirate, en tout point identique au site de votre banque. Dès vos identifiant et mot de passe saisis sur ce faux site, le pirate pourra les récupérer et se rendre avec sur le véritable site de votre banque. À noter qu'il existe d'autres techniques que le DNS spoofing qui permettent de substituer un serveur à un autre, mais elles ne seront pas évoquées ici.



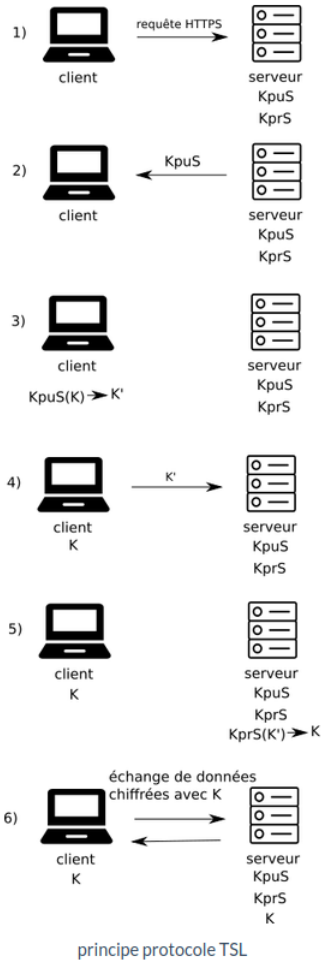
Attaque DNS Spoofing

HTTPS est donc la version sécurisée de HTTP, le but de HTTPS est d'éviter les 2 problèmes évoqués ci-dessus. HTTPS s'appuie sur le protocole TSL (Transport Layer Security) anciennement connu sous le nom de SSL (Secure Sockets Layer). Comment chiffrer les données circulant entre le client et le serveur?

Les communications vont être chiffrées grâce à une clé symétrique. Problème : comment échanger cette clé entre le client et le serveur? Simplement en utilisant une paire clé publique / clé privée! Voici le déroulement des opérations:

- le client effectue une requête HTTPS vers le serveur, en retour le serveur envoie sa clé publique (KpuS) au client
- le client "fabrique" une clé K (qui sera utilisé pour chiffrer les futurs échanges), chiffre cette clé K avec KpuS et envoie la version chiffrée de la clé K au serveur
- le serveur reçoit la version chiffrée de la clé K et la déchiffre en utilisant sa clé privée (KprS). À partir de ce moment-là, le client et le serveur sont en possession de la clé K
- le client et le serveur commencent à échanger des données en les chiffrant et en les déchiffrant à l'aide de la clé K (chiffrement symétrique).

On peut résumer ce processus avec le schéma suivant:



principe protocole TSL

Ce processus se répète à chaque fois qu'un nouveau client effectue une requête HTTPS vers le serveur. Comment éviter les conséquences fâcheuses d'une attaque de type DNS Spoofing?

Pour éviter tout problème, il faut que le serveur puisse justifier de son "identité" ("voici la preuve que je suis bien le site de la banque B et pas un site "pirate""). Pour ce faire, chaque site désirant proposer des transactions HTTPS doit, périodiquement, demander (acheter dans la plupart des cas) un certificat d'authentification (sorte de carte d'identité pour un site internet) auprès d'une autorité habilitée à fournir ce genre de certificats (chaque navigateur web possède une liste des autorités dont il accepte les certificats). Comme dit plus haut, ce certificat permet au site de prouver son "identité" auprès des clients. Nous n'allons pas entrer dans les détails du fonctionnement de ces certificats, mais vous devez juste savoir que le serveur envoie ce certificat au client en même temps que sa clé publique (étape 2 du schéma ci-dessus). En cas d'absence de certificat (ou d'envoi de certificat non conforme), le client stoppe immédiatement les échanges avec le serveur. Il peut arriver de temps en temps que le responsable d'un site oublie de renouveler son certificat à temps (dépasse la date d'expiration), dans ce cas, le navigateur web côté client affichera une page de mise en garde avec un message du style "ATTENTION le certificat d'authentification du site XXX a expiré, il serait prudent de ne pas poursuivre vos échanges avec le site XXXX".