CY Cergy-Paris Université **Enseignant : Kévin Carrier**

Sécurité des Réseaux - Cybersécurité kevin.carrier@cyu.fr

TP : Certificats, Scapy, Sniffing, Attaque DDoS, ARP spoofing (Man-In-The-Middle)

I/ Les certificats.

Une **Autorité de Certification** (CA pour *Certificate Authority* en anglais) est un tiers de confiance permettant d'authentifier l'identité des correspondants. Une autorité de certification délivre des **certificats** décrivant des identités numériques et met à disposition les moyens de vérifier la validité des certificats qu'elle a fournis.

Remarque. *L'autorité de certification (AC) opère elle-même ou peut déléguer l'hébergement de la clé privée du certificat à un opérateur de certification (OC) ou autorité de dépôt. L'AC contrôle et audite l'opérateur de certification sur la base des procédures établies dans la Déclaration des Pratiques de Certification. L'AC est accréditée par une autorité de gestion de la politique qui lui permet d'utiliser un certificat renforcé utilisé par l'OC pour signer la clé publique selon le principe de la signature numérique.*

Les services des autorités de certification sont principalement utilisés dans le cadre de la sécurisation des communications numériques via le protocole **Transport Layer Security (TLS)** utilisé par exemple pour sécuriser les communications web (HTTPS) ou email (SMTP, POP3, IMAP... sur TLS), ainsi que pour la sécurisation des documents numériques (par exemple au moyen de signatures électroniques avancées telles que PAdES pour des documents PDF, ou via le protocole S/MIME pour les emails).

Certains navigateurs web modernes intègrent nativement une liste de certificats provenant de différentes Autorités de Certification choisies selon des règles internes définies par les développeurs du navigateur.

*Q1. Retrouver cette liste de certificats dans votre navigateur (appelé **magasin de certificats**).*

Lorsqu'une personne physique ou morale souhaite mettre en place un serveur web utilisant une communication HTTPS sécurisée par TLS, elle génère une clé publique, une clé privée puis envoie à l'une de ces autorités de certification une demande de signature de certificat (CSR pour *Certificate Signing Request* en anglais) contenant sa clé publique ainsi que des informations sur son identité (coordonnées postales, téléphoniques, email...).

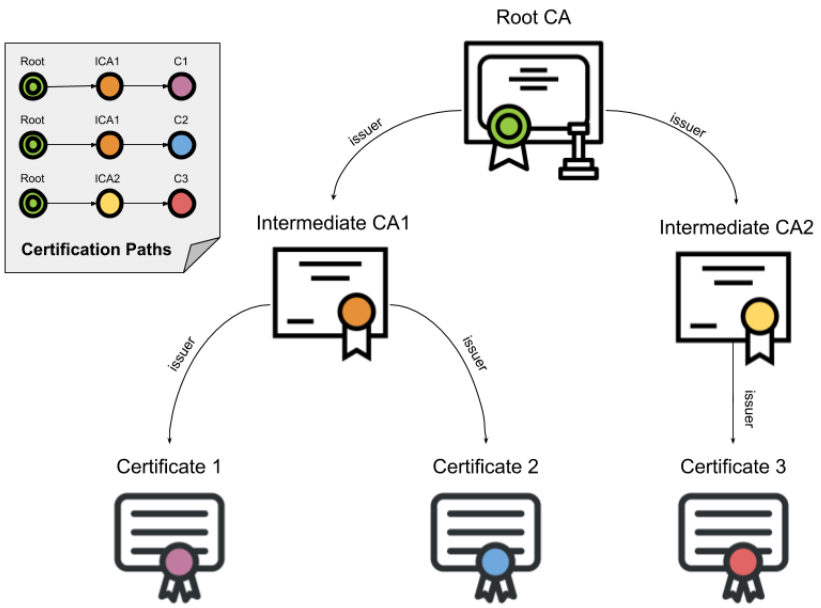
Après vérification de l'identité du demandeur du certificat par une autorité d'enregistrement (RA), l'Autorité de Certification signe le CSR grâce à sa propre clé privée (et non pas avec la clé privée de la personne donc) qui devient alors un certificat puis le transmet en retour à la personne qui en a fait la demande.

Le certificat ainsi retourné sous forme de fichier informatique est intégré dans le serveur web du demandeur. Lorsqu'un utilisateur se connecte à ce serveur web, ce serveur lui transmet à son tour le certificat fourni précédemment par l'Autorité de Certification.

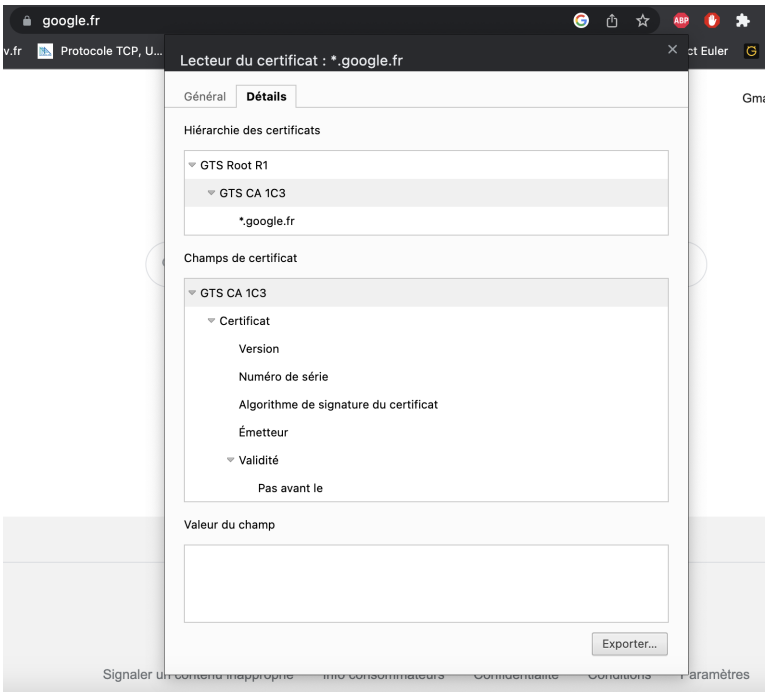
Le navigateur web du client authentifie le certificat du serveur grâce au certificat de l'Autorité de Certification (intégré nativement dans certains navigateurs) qui l'a signé précédemment. L'identité du serveur est ainsi confirmée à l'utilisateur par l'Autorité de Certification.

Le navigateur web contacte ensuite l'Autorité de Certification concernée pour savoir si le certificat du serveur n'a pas été révoqué (= invalidé) depuis qu'il a été émis par l'Autorité de Certification via une demande OCSP.

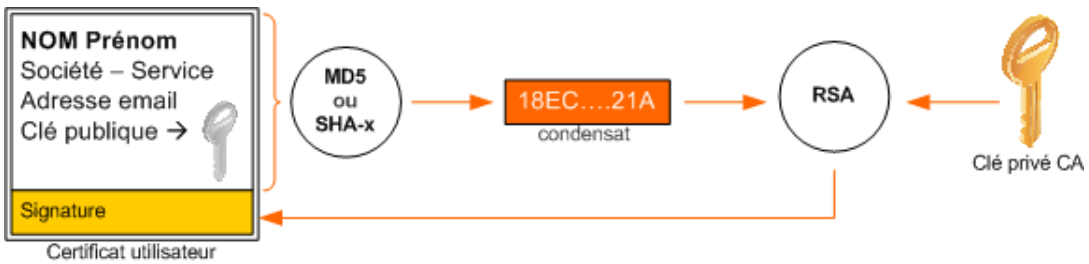
L'autorité de certification qui a délivré le certificat peut elle-même posséder un certificat qui a été émis par une autre autorité de certification. On parle alors de **chaine de certification**.



Q2. Visualiser la chaine de certification de `google.fr` *sur votre navigateur. Quelle est l'autorité de certification* ***racine*** *(ou* root *en anglais)? Quelle est la particularité de cette autorité de certification?*



Q3. On comprend à présent que les certificats fonctionnent sur la ***confiance*** *que l'on peut porter à une autorité de certification (notamment l'autorité* root*). Mais que contiennent exactement ces certificats et quelle mécanisme permet de leur faire confiance?*



Propriété. *Un certificat doit être :*

- ***infalsifiable*** *: la combinaison d'une fonction de hachage et d'une signature numérique empêcher toute modification.*
- ***nominatif*** *: il est délivré à une entité (comme la carte d'identité est délivrée à une personne et une seule).*
- ***certifié*** *: il y a le « tampon » de l'autorité qui l'a délivré (signé par la CA).*

Les certificats permettent de contrôler une **Infrastructure de clés publiques** (PKI pour *Public Key Infrastructure* en anglais). Ils permettent de vérifier l'identité du propriétaire d'une clé publique ce qui permettra par la suite (via le protocole TLS) de mettre en place des communications chiffrées et signées.

Les certificats électroniques respectent des standards spécifiant leur contenu de façon rigoureuse. Les deux formats les plus utilisés aujourd'hui sont :
- X.509, défini dans la RFC 52804 ← *celui que l'on va utiliser dans ce TP* ;
- OpenPGP, défini dans la RFC 48805.

La différence notable entre ces deux formats est qu'un certificat X.509 ne peut contenir qu'un seul identifiant, que cet identifiant doit contenir de nombreux champs prédéfinis, et ne peut être signé que par une seule autorité de certification. Un certificat OpenPGP peut contenir plusieurs identifiants, lesquels autorisent une certaine souplesse sur leur contenu, et peuvent être signés par une multitude d'autres certificats OpenPGP, ce qui permet alors de construire des toiles de confiance.

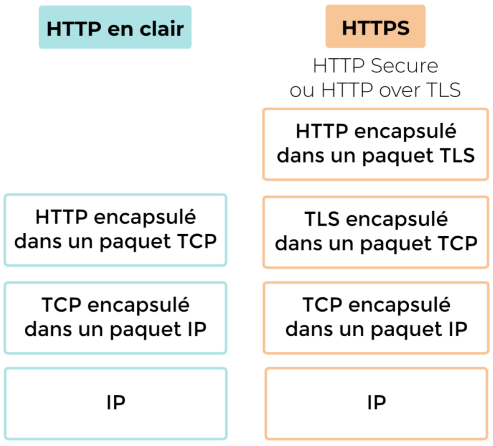
II/ HTTPS : HTTP over TLS

Le petit cadenas vert sur la barre d'adresse de votre navigateur signifie que la connexion est sécurisée avec le protocole HTTPS utilisant TLS (*Transport Layer Security*) (successeur de SSL).

TLS est un protocole cryptographique très utilisé pour sécuriser les connexions réseaux. Il permet de garantir :
- l'authentification du serveur avec les certificats signés par une autorité de certification,
- l'échange d'une clé de session (chiffrement asymétrique),
- la confidentialité des données (chiffrement symétrique),
- l'intégrité des données (MAC : Message Authentication Code).

L'usage le plus courant de TLS est en association avec HTTP pour former le protocole HTTPS (HTTP over TLS).
Le protocole HTTP est un protocole texte réseau encapsulé dans la suite de protocoles Internet TCP/IP de la manière suivante : IP(TCP(HTTP)). Le protocole HTTP n'applique aucune sécurité : les données sont envoyées en clair et sans contrôle d'intégrité. Les données échangées en HTTP peuvent être lues et modifiées par n'importe qui sur le réseau Internet.

HTTPS authentifie et chiffre les données HTTP avec TLS. En HTTPS, le protocole TLS se place entre les protocoles TCP et HTTP comme ceci : IP(TCP(TLS(HTTP))). Ainsi, toutes les données HTTP sont chiffrées et authentifiées, en particulier l'URL, les en-têtes HTTP, et le corps des requêtes et réponses HTTP.



Attention, les données échangées au niveau des protocoles IP et TCP ne sont pas sécurisées par TLS, en particulier les adresses IP du client et du serveur. De plus, les requêtes DNS ne sont pas sécurisées par TLS et peuvent donc fournir des informations sur le nom d'hôte (ou nom de domaine) du serveur.

Remarque. *Les différentes versions de SSL/TLS sont :*

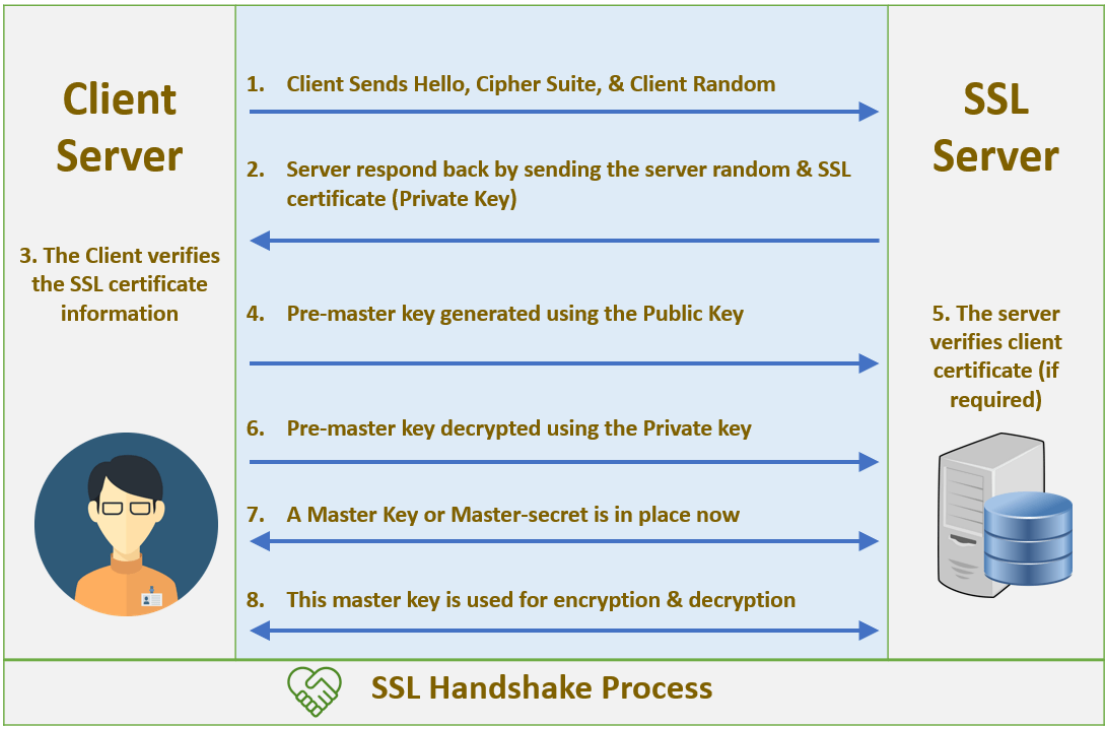
- *SSL : obsolète et considéré non sécurisé.*
- *TLS 1.2 : la plus utilisée.*
- *TLS 1.3 : une suite crypto plus complète mais version encore peu utilisée.*

Une communication TLS se passe en deux temps :

- l'établissement de la liaison (TLS handshake) qui utilise la cryptographie à clé publique (chiffrement asymétrique et signature de certificat) ;
- l'échange de données (TLS record) qui utilise la cryptographie symétrique (chiffrement symétrique et code MAC).

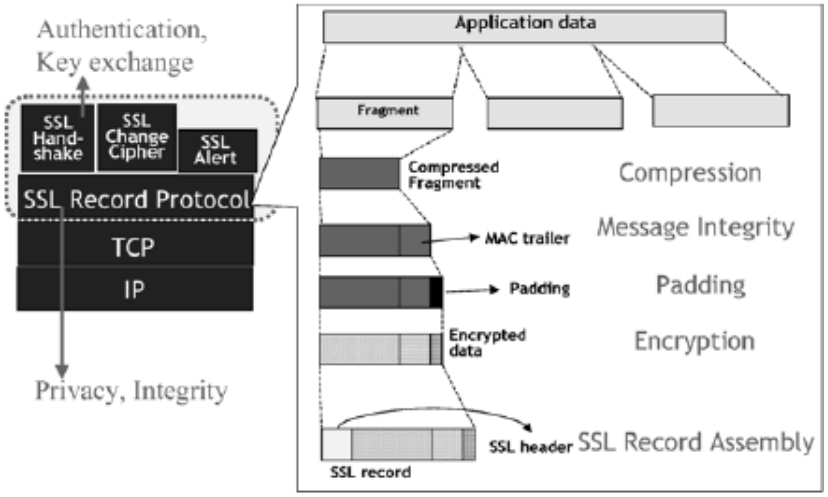
TLS handshake.
Le protocole *TLS handshake* s'effectue une fois au début de la communication. Il utilise la cryptographie asymétrique pour échanger une clé de session.
Avant la communication, le client fait confiance au certificat d'un CA racine et le serveur possède un certificat signé par le même CA racine. C'est la PKI HTTPS.

1. Le client démarre la connexion avec une requête "Client Hello" en clair contenant les versions de TLS et les algorithmes de chiffrement qu'il est capable d'utiliser.
2. Le serveur répond avec une requête "Server Hello" en clair contenant la version de TLS choisie parmi celles proposées par le client, ainsi que son certificat X509 contenant sa clé publique et signée par un CA racine.
3. Le client vérifie la signature du certificat avec la clé publique du CA racine. Si le certificat est expiré ou n'est pas signé par un CA de confiance, le navigateur affiche une erreur. Si le certificat est validé, le client génère une clé secrète symétrique (appelée clé de session) avec un PRNG et de l'entropie.
4. Le client chiffre la clé de session avec la clé publique du serveur et la lui transmet.
6. Le serveur déchiffre la clé de session avec sa propre clé privée.
7. Le client et le serveur partagent maintenant une clé de session symétrique.



TLS record.

1. Le client chiffre la requête HTTP avec la clé de session et calcule le code MAC. Il envoie cette requête chiffrée au serveur.
2. Le serveur déchiffre la requête avec la clé de session et vérifie le code MAC.
3. Il traite la requête HTTP, puis il chiffre la réponse HTTP avec la clé de session et calcule le code MAC.
4. Il envoie cette réponse chiffrée au client qui la déchiffre à son tour et affiche le site Internet. Si le client envoie une autre requête HTTP, il répète les opérations.



Q4. Installez Apache :

```
sudo apt-get install apache2
```

Q5. Créez une paire de clés et un certificat auto-signé avec OpenSSL :

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048
      -keyout /etc/ssl/private/apache-autosigne.key
      -out /etc/ssl/certs/apache-autosigne.crt
```

où

- *openssl req -x509. Demande à OpenSSL de créer un certificat de type x509 auto-signé ;*
- *nodes. Option pour ne pas chiffrer la clé privée avec un mot de passe, pour qu'Apache puisse utiliser la clé privée sans intervention utilisateur ;*
- *days 365. Durée de validité du certificat, d'un an ;*
- *newkey rsa:2048. Option pour générer une nouvelle paire de clés pour le certificat, en utilisant le système RSA avec une clé de 2 048 bits ;*
- *keyout /etc/ssl/private/apache-autosigne.key. Emplacement où sera stockée la clé privée générée ;*
- *out /etc/ssl/certs/apache-autosigne.crt. Emplacement où sera stocké le certificat, qui contient la clé publique générée.*

On pourra laisser les informations par défaut demandées par OpenSSL lors de la génération du certificat.

Q6. Maintenant que le certificat est créé, il faut dire au serveur Apache de l'utiliser. Ouvrir le fichier de configuration du Virtual Host TLS par défaut d'Apache :

```
sudo gedit /etc/apache2/sites-available/default-ssl.conf
```

Remplacer les lignes :

```
SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

par le certificat et la clé privé que vous venez de créer :

```
SSLCertificateFile      /etc/ssl/certs/apache-autosigne.crt
SSLCertificateKeyFile /etc/ssl/private/apache-autosigne.key
```

Remarque. Noter que l'extension .pem ou .crt n'a pas d'importance. L'extension .crt permet d'afficher une version graphique du certificat en double cliquant sur le fichier.

Q7. Activer le mode SSL d'Apache :

```
sudo a2enmod ssl
```

Q8. Activez le Virtual Host TLS d'Apache :

```
sudo a2ensite default-ssl
```

Q9. Et enfin redémarrez Apache :

```
sudo systemctl restart apache2
```

Q10. Visiter la page web : https://localhost dans la barre d'adresse de votre navigateur

Il peut arriver que le navigateur affiche un message d'erreur. Certains navigateurs n'accepte pas les certificats auto-signés. Ajouter une exception pour ce site : cliquer sur Avancé puis Ajouter une exception puis Confirmer l'exception.

Q11. Utiliser Wireshark pour visualiser les TLS handshake et TLS record. Observer les briques cryptographiques.

III/ L'attaque de l'homme du milieu.

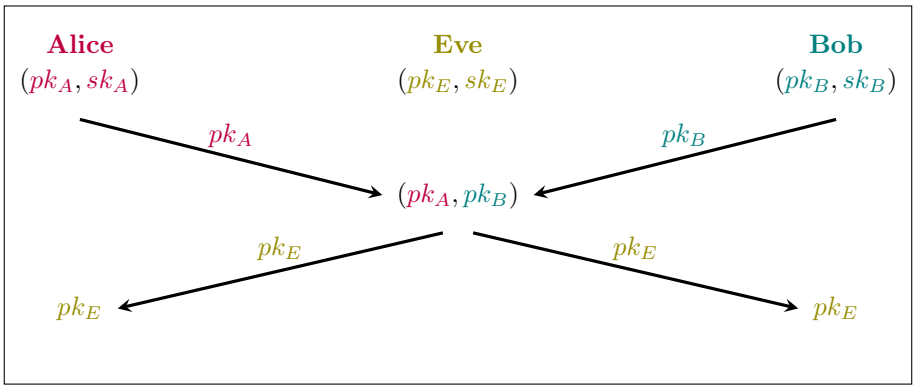
Alice et Bob veulent communiquer de manière sécurisée $\longrightarrow$ Confidentialité + Intégrité + Non-répudiation.

Dans la suite, on suppose que Alice, Bob et Eve ont chacun un couple clé publique/clé privée que l'on notera respectivement pk_A/sk_A, pk_B/sk_B, pk_E/sk_E.

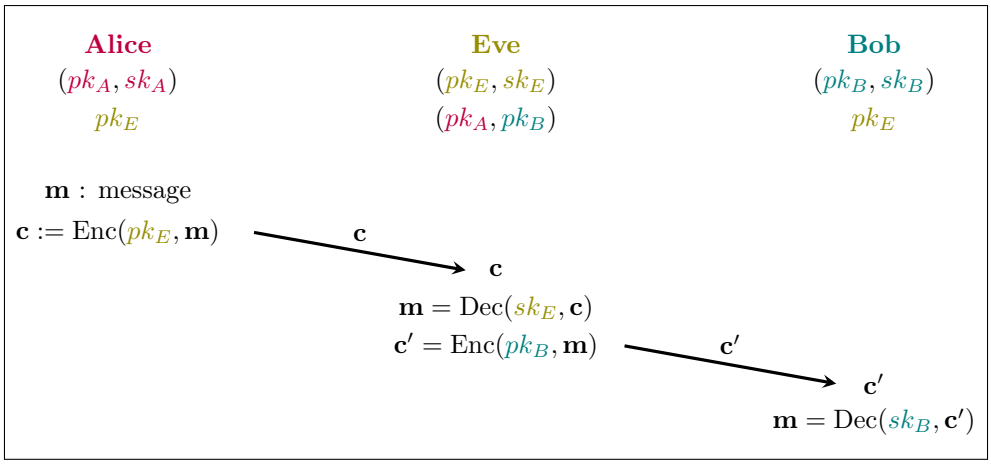
Dans un fonctionnement normal, Alice envoie pk_A à Bob. Bob envoie pk_B à Alice.
$\Longrightarrow$ Chacun peut alors chiffrer des messages avec la clé publique de l'autre ou signer des messages avec sa propre clé privée.

Le principe de l'attaque de l'homme du milieu :

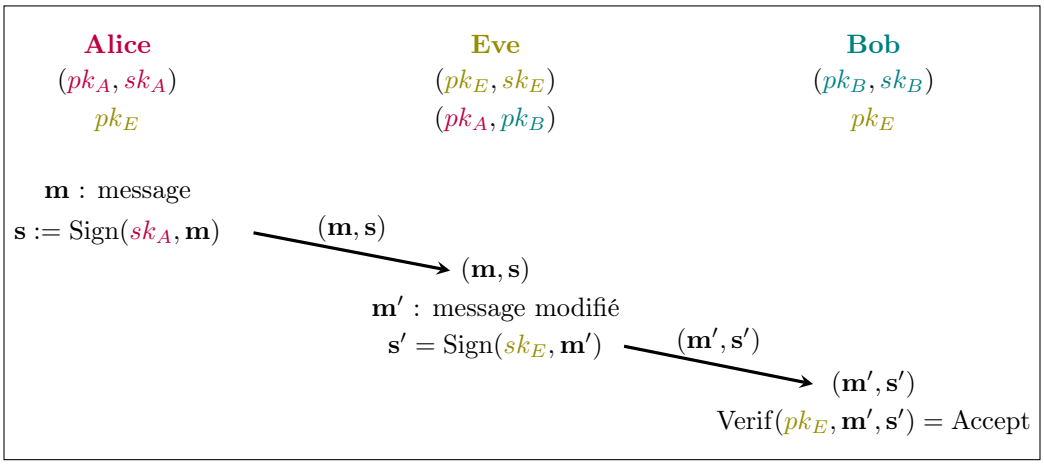
1. L'attaquante Eve se place entre Alice et Bob. Pour cela, elle va **corrompre la table ARP de Alice et de Bob** (cf. plus loin) ;
2. Eve intercepte pk_A destinée à Bob et envoie sa propre clé publique pk_E à la place en faisant croire que c'est bien la clé publique d'Alice ;
3. Idem pour la clé publique de Bob.



À ce moment là, **la confidentialité est compromise**... Lorsque Alice veut envoyer un message à Bob, elle le chiffre avec pk_E en pensant que c'est la clé publique de Bob. Eve intercepte le message et le déchiffre avec sa clé privée sk_E. Elle peut alors lire le contenue du message avant de le renvoyer à Bob en le chiffrant bien cette fois avec la clé pk_B.



L'intégrité et la non-répudiation sont également compromises. Lorsque Alice veut signer un message destiné à Bob, elle produit une signature avec sa clé privée sk_A. Cependant, Eve peut modifier le message puis l'envoyer à Bob en signant le nouveau message avec sa clé privée sk_E. Comme Bob pense que pk_E est la clé publique d'Alice, il va vérifier avec succès la signature.



Corrompre la table ARP.

En fait, toute la partie ci-dessus sur les clés publiques et privées est relativement transparente dans l'attaque que nous allons mettre en place (cachée dans le protocole HTTPS). La partie importante pour nous est de placer l'attaquante Eve entre Alice et Bob au niveau de la couche physique puis réseau.

Commençons par la couche physique. Concrètement, pour mettre en place une attaque de l'homme du milieu au niveau de la couche physique, nous allons corrompre la table ARP qui associe les adresses réseaux (IP) avec les adresses physiques (MAC). Cette attaque est appelée ARP spoofing.

Pour cette partie, on aura besoin de **3 machines** :

- un client : 192.168.1.11/24 (avec un navigateur) ;
- un serveur : 192.168.1.21/24 (serveur Web apache) ;
- un attaquant : 192.168.1.31/24 (Kali Linux).

Si vous utilisez des VM, placez les trois machines dans un réseau NAT et désactivez le mode promiscuité. En réseau NAT, vous pouvez activer le DHCP pour attribuer les adresses IP automatiquement. En outre, vérifiez que chacune de vos machines ont une **adresse MAC différente**.

Remarque. *Pour mettre le clavier en AZERTY, utiliser une des trois commandes suivantes :*

```
setxkbmap -layout fr
loadkeys fr
sudo apt-get install console-data
```

Q12. Sur le serveur, démarrer le serveur Web apache :

```
service apache2 start
```

Q13. Sur la machine du client, vérifier que l'on accède au serveur :

```
curl -v http://192.168.1.21/ >/dev/null
```

Q13. Vérifier que l'on ne voit pas le trafic entre le client et le serveur depuis la machine de l'attaquant. Lancer la commande suivante puis relancer la commande précédente sur la machine du client :

```
tcpdump -ni eth0 ip -X
```

(ip pour afficher que les paquets IP et -X pour plus de détails) ou utiliser Wireshark.

Q14. Interroger la table ARP et vérifier que la correspondance IP/MAC est celle attendue :

```
arp -an
```

Nous allons utiliser Scapy. Scapy est un module Python. C'est un outil puissant en Cybersécurité qui permet d'écouter le réseau (*sniffing*), de lire des trames de n'importe quelle couche du modèle OSI, de créer des trames pour n'importe quelle couche du modèle OSI, d'envoyer ces trames...

Q15. Installer scapy :

```
pip3 install scapy
import scapy.all as scapy
```

La fonction scapy.lsc() permet de lister les différentes commandes scapy.

Q16. ***Sniffing.*** *Pour vous familiariser avec scapy, implémenter un programme qui affiche l'adresse IP de l'émetteur de tous les paquets qui arrivent sur la machine de l'attaquant.*

Le *sniffing* permet de "renifler" le réseau. On pourrait implémenter un genre de Wireshark simplement avec scapy... Noter que le *sniffing* permet également de faire du filtrage. On peut imaginer implémenter un firewall avec scapy...

Scapy permet également de créer des paquets et de les envoyer sur le réseau. Un attaquant peut alors assez facilement sur-solliciter notre serveur Web...

Q17. Implémenter une ***attaque par déni de service*** *(DoS) avec scapy. L'attaque consiste à sur-solliciter le serveur Web jusqu'à ce qu'il* crash. *Penser à utiliser des threads (parallélisme). Vérifier que votre attaque à bien fonctionné en essayant de vous connecter au serveur Web à partir du client.*

ATTENTION : Pour la question Q17, en fait, ça n'est pas si simple... Le serveur arrive à traiter les requêtes assez vite pour ne pas être submergé... Il faudrait mener cette attaque en parallèle avec plusieurs machines attaquantes (DDoS).

À présent, revenons à nos moutons, à savoir notre attaque Man-In-The-Middle par corruption de la table ARP. La première étape consiste à récupérer les adresses MAC du serveur Web et du client depuis la machine de l'attaquant.

Q18. Écrire une fonction qui permet de récupérer l'adresses MAC d'une IP donnée. Utiliser cette fonction pour récupérer les MAC du client et du serveur Web depuis l'attaquant. Votre fonctionne permet-elle de récupérer votre propre adresse MAC? Pourquoi?

Q19. Depuis la machine de l'attaquant, créer un paquet ARP destiné au serveur qui associe l'IP du client au MAC de l'attaquant. Encapsuler ce paquet ARP dans une trame Ethernet.

Noter que du point de vu du serveur, l'attaquant et le serveur ont la même adresse MAC. Ce n'est pas un problème : ça arrive qu'une même carte réseau porte plusieurs IP (exemple : routeur d'un LAN).

Q20. Il faut également corrompre la table ARP du client. Depuis la machine de l'attaquant, créer le paquet Ethernet/ARP à destination du client.

Q21. La table ARP est mise à jour régulièrement! Il faut donc envoyer ces paquets en continue. Utiliser la fonction srploop ou plus simplement une boucle `while(True)` pour envoyer vos trames Ethernet toutes les secondes.

Que ce passe-t-il? L'attaquant intercepte les paquets au niveau de la couche physique (Ethernet), les paquets lui sont bien adressés mais au niveau Réseau (IP), non. Donc au niveau de la pile IP, le paquet est jeté.

Q22. Proposer un moyen pour faire suivre les paquets (soit avec scapy, ou plus simplement en activant le mode routeur de la machine de l'attaquant). Tester votre solution. Le client doit pouvoir se connecter au site Web du serveur en toute transparence. De son côté, l'attaquant doit pouvoir observer le trafic entre le client et le serveur (en utilisant netstat, ou tcpdump, ou bien en sniffant le réseau avec scapy, ou encore avec Wireshark).

IV/ Redirection vers le serveur Web de l'attaquant.

À présent, on veut que lorsque le client se connecte au serveur Web, celui-ci soit redirigé vers le serveur Web de l'attaquant.

Q23. Commencer par modifier la page html de l'attaquant pour bien la distinguer de celle du serveur légitime.

```
vi var/www/html/index.html
```

Q24. Utiliser iptables pour rediriger les paquets sur l'attaquant (la réponse est automatiquement redirigée avec iptables).

Q25. Supposons que le serveur filtre les INPUT : il n'accepte que les connections venant du client. L'attaquant veut pouvoir malgré tout se connecter au serveur. Utiliser iptables pour changer la source des paquets par l'IP de l'attaquant (la réponse est automatiquement redirigée avec iptables).

CONCLUSION : Les machines sur un même LAN doivent avoir le même niveau de confiance!

Utiliser un certificat auto-signé (ou expiré) ne diminue pas la force du chiffrement des données HTTP par rapport à un certificat valide signé par une autorité de certification. En revanche, un certificat auto-signé ne permet pas d'authentifier le serveur web, ce qui signifie qu'un attaquant exécutant une attaque MITM peut intercepter et remplacer le certificat du serveur par le sien, puis déchiffrer et modifier toutes les informations échangées entre le client et le serveur, réduisant à néant la sécurité de TLS.

Un certificat auto-signé est généralement utilisé en environnement local de développement, de test ou de préproduction.

Par ailleurs, le certificat racine d'une autorité de certification racine est auto-signé, puisque c'est elle-même qui est chargée de signer les certificats ! Cependant, ce certificat fait partie des certificats de confiance des machines clientes, et est donc reconnu comme valide.

V/ Faire signer ses certificats par une vraie CA

Avant de commencer cette partie, il faut stopper l'attaque ARP précédentes et réinitialiser la table ARP corrompue du client avec la commande arp.

Plutôt que d'autosigner votre certificat, vous pouvez le faire signer par un tiers de confiance : une autorité de certification comme *Let's Encrypt*.

`https://letsencrypt.org/fr/`

ATTENTION : Pour utiliser letsencrypt, il faut fournir des informations que nous n'avons pas comme un DNS.

Ici, on va simplifier les choses en créant notre propre autorité de certification.

Q26. Créer votre propre autorité de certification à l'aide de openssl.

Q27. Faire reconnaitre votre autorité de certification par le navigateur du client.

Q28. Créer le certificat du serveur Web avec openssl.

Q29. Installer le certificat sur le serveur Web. Pour cela, copiez le certificat et sa clé sur le serveur Web et mettez à jour la configuration d'apache pour utiliser ce nouveau certificat (cf. début du TP). Connectez-vous avec le client et vérifiez que le certificat est accepté automatiquement.

Q30. Retenter un Man-In-The-Middle (ARP spoofing) sur le serveur maintenant qu'il est protégé par un certificat digne de ce nom!

CONCLUSION : Dans le cadre d'un déploiement de services SSL en entreprise (serveurs web, LDAP, VPN...), l'entreprise doit obtenir des certificats d'une autorité reconnue par défaut, soit créer sa propre autorité. Dans le second cas, le certificat de l'autorité locale peut par exemple être installé lors de la configuration initiale des postes de travail.
Retenez bien qu'il existe un risque réel de sécurité lors de l'acceptation de certificats auto-signés, non reconnus automatiquement par le navigateur web.