ARCHITECTURES MATÉRIELLES ET REPRÉSENTATION DES DONNÉES

NSI 04

SÉCURISATION DES DONNÉES

[Frédéric Peurière – Marion Szpieg]

1. Rappels de première

Machine cliente	Machine serveur
4. Protocoles : HTTP, HTTPS, DNS, DHCP …	**4.** Protocoles : HTTP, HTTPS, DNS, DHCP …
3. Protocoles : TCP, UDP ...	**3.** Protocoles : TCP, UDP ...
2. Protocoles : IPV4, IPV6 ...	**2.** Protocoles : IPV4, IPV6 ...
1. Protocoles : ARP, ETHERNET II ...	**1.** Protocoles : ARP, ETHERNET II ...

Nous avons vu l'an dernier que lorsqu'une machine cliente et une machine serveur communiquaient, chaque couche du modèle TCP/IP de la machine cliente échange avec la couche correspondante chez la machine serveur.

Lorsque deux machines discutent via le protocole http, les données transitent sur le réseau en toute transparence. Cela peut causer différents problèmes :

- si des paquets sont interceptés par une 3^{e} machine, il suffira de les lire pour récupérer le message
- si les paquets sont interceptés et modifiés, la machine destinataire ne s'en rendra pas compte
- etc...

Pour éviter ces problèmes, il a fallu introduire de nouveaux protocoles afin de sécuriser les communications sur un réseau.

2. Protocoles de sécurisation : la sous-couche SSL et TLS

Machine cliente	Machine serveur
4. Protocole : HTTPS	**4.** Protocole : HTTPS
3. **Protocoles : SSL, TLS** / Protocoles : TCP, UDP ...	**3.** **Protocoles : SSL, TLS** / Protocoles : TCP, UDP ...
2. Protocoles : IPV4, IPV6 ...	**2.** Protocoles : IPV4, IPV6 ...
1. Protocoles : ARP, ETHERNET II ...	**1.** Protocoles : ARP, ETHERNET II ...

Pour sécuriser les données échangées via un réseau, les protocoles SSL (Secure Sockets Layer) puis TLS (Transport Layer Security) ont été inventés et améliorés depuis 1994. Ces protocoles sécurisent seulement les informations de la couche 4 : celles des couches 3, 2 et 1 ne sont pas concernées (sinon, les paquets auraient des difficultés à arriver à destination!).

Après ajout de cette sous-couche SSL/TLS, http devient https et les serveurs web qui attendaient les connexions non sécurisées sur le port ……. les attendent désormais sécurisées sur le port …….. .

3. Différents éléments essentiels de la sécurisation

3.1. Intégrité : comment être sûr que le message reçu n'a pas été modifié par rapport à sa version originale ?

Pour garantir l'intégrité de données qu'on souhaite envoyer sur un réseau, on utilise un **algorithme de hachage**. Le principe : on applique l'algorithme sur le document qu'on souhaite envoyer et l'algorithme calcule une empreinte correspondant au document. Voici quelques points fondamentaux du fonctionnement des algorithmes de hachage :

Exemples d'algorithmes de hachage les plus utilisés :

- MD5 (Message Digest Version 5) dont les empreintes font 128 bits, soit 16 octets
- SHA (Secure Hash Algorithm) avec des empreintes de tailles différentes en fonction des versions

Exemple d'utilisation de l'algorithme de hachage SHA dans un terminal Linux (utilisation de la commande : openssl dgst [algorithme voulu] NomDuFichier) :

```
(base) marion@marion-Inspiron-7501:~$ echo "message top secret" > messagetop.txt
(base) marion@marion-Inspiron-7501:~$ cat messagetop.txt
message top secret
(base) marion@marion-Inspiron-7501:~$ openssl dgst -sha1 messagetop.txt
SHA1(messagetop.txt)= 3d52fb6943868fa53353d4de9a6c0dea92591124
(base) marion@marion-Inspiron-7501:~$ openssl dgst -sha1 messagetop.txt
SHA1(messagetop.txt)= 3d52fb6943868fa53353d4de9a6c0dea92591124
(base) marion@marion-Inspiron-7501:~$ echo "message tip secret" > messagetip.txt
(base) marion@marion-Inspiron-7501:~$ cat messagetip.txt
message tip secret
(base) marion@marion-Inspiron-7501:~$ openssl dgst -sha1 messagetip.txt
SHA1(messagetip.txt)= 9192bf4ac66b56d467c3cee384b91c2c5062be0b
```

Remarque : dans un terminal windows, vous pouvez faire la même chose en remplaçant « openssl dgst -sha1 » par « Get-FileHash -Algorithm SHA1 »

```
PS C:\Users\Marion> Get-FileHash -Algorithm SHA1 .\messagetop.txt

Algorithm       Hash
---------       ----
SHA1            3D52FB6943868FA53353D4DE9A6C0DEA92591124


PS C:\Users\Marion> Get-FileHash -Algorithm SHA1 .\messagetip.txt

Algorithm       Hash
---------       ----
SHA1            9192BF4AC66B56D467C3CEE384B91C2C5062BE0B
```

Cas concrets de l’utilisation des algorithmes de hachage :

3.2. Confidentialité : comment « cacher » les données qu’on envoie ?

Pour éviter qu’un serveur pirate ou qu’une personne mal intentionnée intercepte les données qu’on envoie sur un réseau et les lise, on **chiffre** les données avant de les envoyer. Si les données chiffrées sont interceptées, elles ne seront pas lisibles par la personne/le serveur qui les a interceptées.

Problème : comment faire pour arriver à chiffrer les données et faire en sorte que seuls l’expéditeur et le destinataire puissent les déchiffrer ?

L’algorithme de chiffrement symétrique le plus utilisé actuellement est l’algorithme AES (Advanced Encryption Standard). Les clés utilisées par cet algorithme font 128, 192 ou 256 bits.

Exemple d’utilisation sous Linux :

```
(base) marion@marion-Inspiron-7501:~$ echo message top secret > messagetop.txt
(base) marion@marion-Inspiron-7501:~$ cat messagetop.txt
message top secret
(base) marion@marion-Inspiron-7501:~$ openssl enc -nosalt -aes-256-ecb -in messagetop.txt -out
 messagetopc -base64 -K 01234567890123456789012345678901234567890123456789012345678901234567890123
(base) marion@marion-Inspiron-7501:~$ cat messagetopc
FNjsoK9pbrSSAQbHJJ3XTgBb7RTK1EdACZYHM7VjQAI=
(base) marion@marion-Inspiron-7501:~$ openssl enc -d -nosalt -aes-256-ecb -in messagetopc -out
 messagetopcd -base64 -K 01234567890123456789012345678901234567890123456789012345678901234567890123
(base) marion@marion-Inspiron-7501:~$ cat messagetopcd
message top secret
```

Remarque sur les différents paramètres :

- « enc » : pour chiffrer
- « enc -d » : pour déchiffrer
- « -nosalt » : pour ne pas rajouter de « grain de sel »
- « -aes-256-ecb » : algorithme de chiffrement choisi
- « -in » : pour indiquer le document à chiffrer
- « -out » : pour indiquer le document de sortie où stocker le document chiffré
- « -base64 » : pour être sûr d’avoir des caractères affichables dans le document de sortie
- « -K ... » : la clé choisie, quoi doit faire une longueur de 64 caractères

3.3. Authentification : comment être sûr de l'identité du serveur sollicité ?

Lors d'un échange entre une machine cliente et une machine serveur, il est possible qu'un pirate se fasse passer pour la machine serveur afin de récupérer des données sensibles. Par exemple, si vous consultez vos comptes bancaires et que le site sur lequel vous entrez votre identifiant et votre mot de passe est un site pirate identique en tout point au site de votre banque habituel, votre identifiant et votre mot de passe sont récupérés par le serveur pirate, qui peut alors faire des transferts d'argent à partir de votre compte bancaire.

Pour éviter ce problème, on assure l'authentification en deux étapes : on utilise d'un **algorithme de chiffrement asymétrique** et on fait appel à **un tiers de confiance appelé centre certificateur**.

Principe de fonctionnement des algorithmes de chiffrement asymétrique.

Ces algorithmes utilisent cette fois-ci deux clés de chiffrement : une clé privée que seule la machine serveur connaît (et garde secrètement) et une clé publique qui peut être connue de tout le monde.

Voici quelques points fondamentaux du fonctionnement des algorithmes de chiffrement asymétrique :

L'algorithme de chiffrement asymétrique le plus utilisé actuellement est l'algorithme RSA (Rivest Shamir Adleman, noms des 3 mathématiciens l'ayant inventé). Il est très utilisé dans le e-commerce, car très fiable et impossible à « casser » en temps raisonnable.

Cet algorithme repose sur l'utilisation de très très grands nombres premiers et du fait qu'actuellement on ne sait pas décomposer en temps raisonnable de très grands nombres en produit de facteurs premiers. Si l'ordinateur quantique venait à exister, le système RSA n'y résisterait pas, ce qui poserait de gros problèmes notamment pour les achats en ligne.

Exemple d'utilisation sous Linux :

1. **Création de la clé privée :**

> openssl genrsa -out privkey.pem 2048

Remarques paramètres :

- « genrsa » : pour générer une clé privée avec l'algorithme rsa
- « -out » + nom du fichier de sortie
- 2048 : longueur de la clé RSA voulue

```
(base) marion@marion-Inspiron-7501:~$ openssl genrsa -out privkey.pem 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.......+++++
.....................................................................++
+++
e is 65537 (0x010001)
(base) marion@marion-Inspiron-7501:~$ cat privkey.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEAs7FOokJzdLd4ig7BlzrYwVXRXbI70bkTx9YOa97eykRCanM3
d4r4J3OxuaqYhh+7nt5xakpQK7yrejYVZFw2aY3heJSdt6QNfh6IYVvjjfofQFyI
rF2dfmyDklFUHQ9K+wKeNRsqlk66lLdwvhdfob4pINzfVOjv8++rK0XefmEy7J2w
5UX+QGFuIQtIMHp2YgZQX4iwyRF9oyZMaQw1fLooPpEOzjB+DUuHWv1RSlbyHvYW
eRqHzu9FC3kKEboAMwCAYDCNWMzYkKQYsPQCBGXPKzm3f0sJFXVhgOqfBccmcxjj
3Vlrddh1oK9GoEN1O5EEhiCXIYZMKGxb9nQ/4wIDAQABAoIBAQCCk+4Ya0GLJ2go
quBzDQhGNh6Wf9aIaH2/UgXYzXQId+HriR5deFcsFB6qVRXRIW4t3Gd+XdI0dj4F
6RdSfs/rlHasMRMLR+OmKnnghe2uVnwvQQ7wcnboT+KsiP0ewOYz0Ys1/C/OWher
ykHbYRDVcGJixx5jjYIRObChGXy9SJtv6YlphXmnVqc7uo0ZSgSXES2JU92TPkvi
YhmYT8ngrS3as7gNk2wywlfjkYUw/e/QDYNkA6wEdKs82EUOrECjBqVbPwbHfxM5
G68vB3//uMMPK+h9dUpdj9iOz/DD53cLmMyyFkYFhwc0iBgY2KHVcsrEDOBEEjU+
zCrYeXmBAoGBAO1Uy4qMPNhLqGmWeDpEP/AFnystb8aP35HkaY4EcLX/K5zU3zAh
5qUk19Tmtll5RerUCvSCMJI0EG9e9AlqvdMD0xRH3SpyJTDgHXOFXr+naC3gRCa4
Mh0jSF9mWSrCTqNG/QlX1lSv7CLHlMTjYU/0PF0es2peO33JQ6Pe/dA7AoGBAMHT
07vayIo23EnOo7E6wf9S9EIUoeMW8YGb1EnMJwheQVx0It5HbS/I8C2Vo/AE/3qN
i5pzwhhO4SiTmgVja2MoV/5iAp1ky7hqvLL1tOpA0cBifNtAoyLNnGXACwcDkPct
6nDjrIAyhd0g1lJBwqcS120u9vpbje11NlenHTx5AoGBAKSX2S12AsCS//Jya2El
ch45F+Eu+sT/hLK+cUIP7Ecb95glvb+40yeW/KRnePFmfMnWUt6t55eNyhQjCyX4
xDsy/NGk4UqNrSNPWr2CMmrCZCrkzP75Ya/5hzi6V36jETyFf/lZ1IgZFbAtMAy4
EQYNQUvnfdB4F9Qtn95nSSl/AoGBALXjTkJu4Nmy+a3ZWAsslHhBJC4OpowwL5ns
3Z4TEl3r9XIJGd91L6Uz7w8X6auHRSeARI1AK9gbzXcCa/q6fT+j0E9LMiD5QaNh
popZwNhC05Kx2XD2iyBRyn5BZRfD5PnQALzHKk1zG+OSuQyf4UZfjAFNnT77Q4ie
AcznaawhAoGBAM/AHGLmLfVExGASNdp5aXSZCBcDuYNqd8r85laHy2QzTvNDD3Ca
YtMMS2TVmvO+d7le72uolyd7+4UOiElw2oiBMTqj6VcfD5evPYczL1VTP5Nu+fXa
i1MAkV6SWRfuqJxjhuHgjppz0kTzp5bCgXah64mYJFwxMX+a6pQcULDq
-----END RSA PRIVATE KEY-----
```

2. **Création de la clé publique à partir de la clé privée :**

```
(base) marion@marion-Inspiron-7501:~$ openssl rsa -in privkey.pem
-outform PEM -pubout -out pubkey.pem
writing RSA key
(base) marion@marion-Inspiron-7501:~$ cat pubkey.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs7FOokJzdLd4ig7BlzrY
wVXRXbI70bkTx9YOa97eykRCanM3d4r4J3OxuaqYhh+7nt5xakpQK7yrejYVZFw2
aY3heJSdt6QNfh6IYVvjjfofQFyIrF2dfmyDklFUHQ9K+wKeNRsqlk66lLdwvhdf
ob4pINzfVOjv8++rK0XefmEy7J2w5UX+QGFuIQtIMHp2YgZQX4iwyRF9oyZMaQw1
fLooPpEOzjB+DUuHWv1RSlbyHvYWeRqHzu9FC3kKEboAMwCAYDCNWMzYkKQYsPQC
BGXPKzm3f0sJFXVhgOqfBccmcxjj3Vlrddh1oK9GoEN1O5EEhiCXIYZMKGxb9nQ/
4wIDAQAB
-----END PUBLIC KEY-----
```

On remarque que

Remarques paramètres :

- « rsa » : pour indiquer un calcul de clé
- « -in » + nom du fichier contenant la clé privée
- « -outform PEM » : pour indiquer le format du fichier de sortie
- « -pubout » : indique que le résultat du calcul sera une clé publique
- « -out » + nom du fichier de sortie

3. **Chiffrement du document avec la clé publique :**

```
(base) marion@marion-Inspiron-7501:~$ openssl rsautl -encrypt
-inkey pubkey.pem -pubin -in messagetop.txt -out messagetoprsa

(base) marion@marion-Inspiron-7501:~$ cat messagetoprsa
[2Xreyyy
```

4. **Déchiffrement du document avec la clé privée :**

```
(base) marion@marion-Inspiron-7501:~$ openssl rsautl -decrypt
-inkey privkey.pem -in messagetoprsa
message top secret
```

Remarques paramètres :

- « rsautl » : pour indiquer qu'on utilise une des fonctions utilitaires liées à l'algorithme rsa
- « -decrypt » : pour indiquer qu'on veut déchiffrer
- « -inkey pubkey.pem » : indique le fichier qui contient la clé de déchiffrement
- « -in » + nom du fichier à déchiffrer
- la sortie se fait sur l'écran (si on la veut dans un fichier, rajouter « -out » + nom du fichier de sortie)

Principe de fonctionnement des certificats électroniques.

Lorsqu'un site (marchand ou bancaire par exemple) crée ses clés privées et publiques, afin de pouvoir prouver son identité aux futurs clients, il passe part un tiers de confiance qui va vérifier physiquement son identité et émettre un certificat électronique. Ce tiers de confiance s'appelle une autorité de certification.

Lorsqu'une autorité de certification délivre un certificat, ce dernier contient :

L'obtention d'un certificat est un service généralement payant (le prix peut varier de 10€ à plusieurs centaines d'euros par an en fonction de la qualité de la vérification préalable de l'identité du demandeur)

Les autorités de certification sont des organismes enregistrés et certifiés par des Etats, des autorités de contrôle de l'Internet ou des entreprises privées connues : Amazon Trust Services, CertEurope, Cisco, DigiCert, Google Trust Services LLC, Gouvernement français (ANSSI, DCSSI), Microsoft, Symantec, Visa…

Des clés publiques des principaux organismes sont installées avec les systèmes d'exploitation ou les navigateurs. On peut y avoir accès via n'importe quel navigateur ou via un terminal avec les commandes suivantes :

- sous Windows avec la commande cd cert
- sous Linux avec ls /etc/ssl/certs
- avec Firefox : options → vie privée et sécurité → Afficher les certificats…
- avec Chrome : paramètres → confidentialité et sécurité → sécurité → gérer les certificats
- avec Safari : ...

Chaque centre certificateur fait signer son certificat par certains autres centres certificateurs. C'est ce qu'on appelle la chaîne de certificat. Ainsi, un serveur pirate qui voudrait se faire passer pour un centre certificateur officiel serait rapidement détecté et son certificat sera rejeté par n'importe quel navigateur internet.

Une autorité de certification officielle qui manquerait à ses obligations de sécurité peut se faire exclure de la liste des centres certificateur par le consortium des autres centres. Dans ce cas, un site qui aurait obtenu son certificat électronique via le centre certificateur rejeté n'aurait plus de connexion officiellement sécurisée. C'est arrivé avec le centre certificateur StartCom en 2020.

4. Fonctionnement (simplifié) du protocole sécurisé https

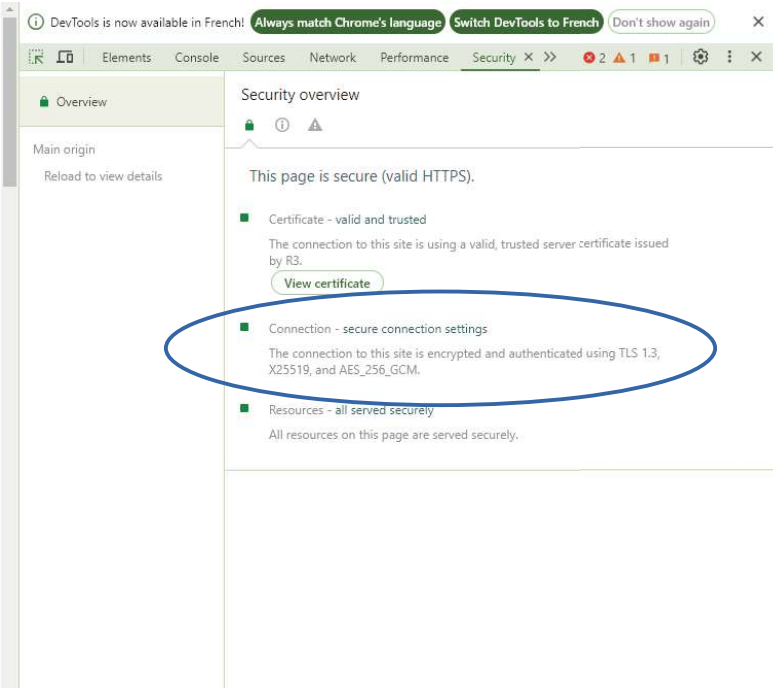
Le protocole HTTPS est un protocole de la couche 4, qui va mettre en œuvre les algorithmes vus dans la partie 3 afin d'assurer la confidentialité, l'authentification et l'intégrité des données échangées.

Il s'appuie donc sur une suite de chiffrements (*cipher suit* en anglais) qui fait appel à un algorithme de hachage, un algorithme de chiffrement symétrique et un algorithme de chiffrement asymétrique.

Voici le fonctionnement plus en détail :

Remarque : pourquoi utiliser l'algorithme de chiffrement symétrique alors que les premiers échanges se font par un algorithme de chiffrement asymétrique ? Car les algorithmes de chiffrement asymétriques demandent plus de temps de calcul que les algorithmes symétriques. Ainsi, lorsqu'on a beaucoup de données à échanger, il vaut mieux utiliser des algorithmes symétriques pour gagner du temps. **MAIS**, on ne peut pas se passer des algorithmes asymétriques afin d'assurer l'authentification du serveur.

Exemple de *cipher suit* du site du lycée (en mars 2024) :



On voit en particulier les algorithmes suivants :

- X25519 : algorithme asymétrique pour l'authentification
- AES : algorithme symétrique pour la confidentialité du flux des données échangées

Pour voir toutes les *cipher suit* disponibles avec openssl, il suffit de taper dans un terminal la commande : openssl ciphers