

Démystification sur les clés cryptographiques

Chiffrement, signature et certificats.

Qui suis je ?

Arnaud JOLLY (chez SOAT depuis 2011)

Pôle WEB - Team Perseverance



Plan

1. Chiffrement (7 slides)
2. Signatures (4 slides)
3. Certificats (5 slides)

Chiffrement

- but: s'assurer que seul le destinataire d'un message puisse lire le message
- on chiffre de la data donc des octets (*bytes*), pas seulement du texte
- toutes les opérations manipulent des octets et renvoient des octets
- le résultat du chiffrement est appelé cryptogramme. Il est donc aussi composé d'octets.

Chiffrer \neq Encoder

Prenons un header HTTP que vous utilisez peut-être dans vos applis 🙈 (💩)

```
Authorization: Basic am9sbHlfYTpQNDVTdzByRF8xMg==
```

Tentons de décoder cette chaine (⚠️ je n'ai pas dit déchiffrer).

Allons voir notre ami [CyberChef](#)

Chiffrement asymétrique 1/2

Ce mode de chiffrement/déchiffrement repose sur une paire de clés.

Ce qui est chiffré avec l'une n'est déchiffrable qu'avec l'autre.

Une des clés est appelée "Public Key", l'autre "Private Key".

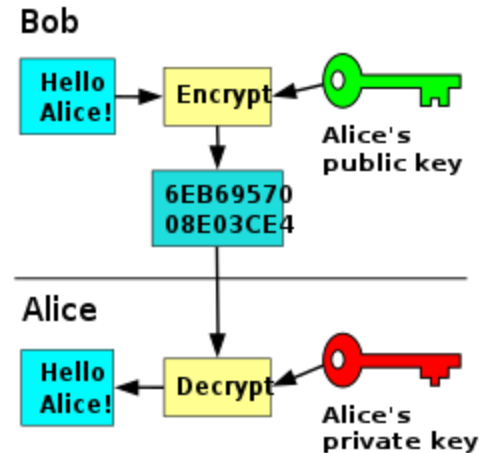
Selon le type de clé (RSA, Elliptic Curve, etc.) la structure même de la clé change mais pas son rôle.

Exemple pour une clé RSA:

```
openssl genrsa [-aes128] -out alice_private.pem 2048  
openssl rsa -in alice_private.pem -pubout -out alice_public.pem  
openssl rsa -in <fichier.pem> -text -noout
```

Chiffrement asymétrique 2/2

Bob veut envoyer un message à Alice et veut être sûr que seule Alice pourra le lire. La clé que seule Alice possède étant sa propre clé privée, si Bob utilise la clé publique d'Alice, il sera sûr que seule cette dernière pourra le lire.



```
echo "vim > emacs!" > top_secret.txt
openssl rsautl -encrypt -inkey alice_public.pem -pubin -in top_secret.txt -out top_secret.txt.enc
xxd top_secret.txt.enc
cp top_secret.txt.enc ../alice
openssl rsautl -decrypt -inkey alice_private.pem -in top_secret.txt.enc -out top_secret.txt
```

Chiffrement symétrique 1/3

L'idée c'est qu'on tire une clé de la taille qui convient par rapport à l'algorithme utilisé et que cette clé va être partagée entre les deux parties.

Avantage:

- algorithmes moins gourmands et plus rapides pour chiffrer/déchiffrer

Inconvénient:

- il y a partage de donnée sensible (la clé)

Chiffrement symétrique 2/3

Exemple avec AES (Advanced Encryption Standard)

C'est un des algorithmes de chiffrement symétrique, si ce n'est le plus répandu. Il a besoin d'une clé d'une certaine taille et d'une configuration initiale appelé "Initialization Vector" (IV).

Ex:

- aes-256 prend des clés de taille 256bits (32bytes) et un IV de 16bytes
- aes-128 prend des clés de taille 128bits (16bytes) et un IV de 16bytes

Comme la taille de clé est fixe, on passe souvent par des fonctions de hashage qui vont hasher une passphrase de la longueur de votre choix pour en sortir une clé de la bonne taille.

Chiffrement symétrique 3/3

Si ça vous intéresse, voici les commandes pour chiffrer/déchiffrer un fichier avec AES.

```
key="0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20"  
iv="0102030405060708090A0B0C0D0E0F10"  
openssl enc -aes-256-cbc -e -in plaintext.txt -out aes-encrypted.enc -K "$key" -iv "$iv"  
openssl enc -aes-256-cbc -d -in aes-encrypted.enc -out decrypted-file.txt -K "$key" -iv "$iv"
```

Et si vous n'avez pas la paire "clé/iv", c'est que ces deux informations vont être dérivés d'un password qui vous sera demandé.

```
openssl enc -aes-256-cbc -e -in plaintext.txt -out aes-encrypted.enc -pbkdf2  
openssl enc -aes-256-cbc -d -in aes-encrypted.enc -out decrypted-file.txt -pbkdf2
```

L'option `-pbkdf2` est là pour préciser la fonction de dérivation que l'on va utiliser au lieu de celle par défaut qui est dépassée mais gardée pour raison de rétrocompatibilité.

Signatures, kezako ?

Si on chiffre avec la clé publique d'une personne, seule cette personne peut lire le contenu.

Si on chiffre avec la clé privée d'une personne, la donnée chiffrée pourra être lu par toute personne ayant sa clé publique (donc tout le monde).

Question: Mais qui a la clé privée d'une personne ?

Réponse: Seulement cette personne

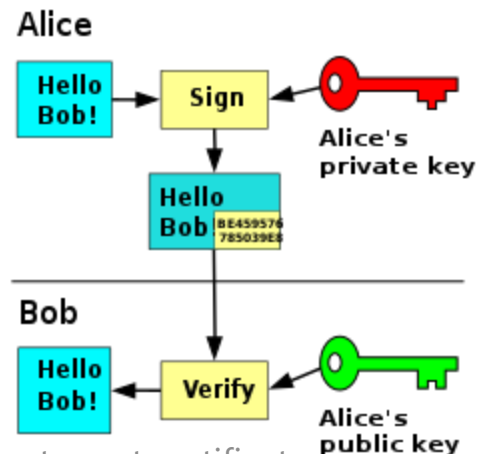
On se sert donc de ce mécanisme pour assurer que le cryptogramme a bien été écrit par la personne qui dit l'avoir envoyé.

Signatures, exploitation

L'idée d'une signature et d'être envoyée à côté d'un document pour que la personne qui a le couple document/signature puisse vérifier que le document est bien le même et n'a pas été altéré par un petit malin de livreur.

Le signataire ne va pas chiffrer le document entier mais un hash de ce document (un md5, un sha1, etc.) ca permet d'avoir des signatures de la même taille peu importe le document (utile si le document est un backup d'1To 🙌).

Alice envoie un message signé à Bob.



Signatures, vérification

Bob souhaite vérifier que le message n'a pas été modifié, on ne sait jamais, le livreur de La Poste lui en veut d'avoir reporté la détérioration de son colis la semaine passée 🤔.

Pour ça, il va:

- recalculer le hash du message envoyé par Alice
- déchiffrer la signature accompagnant le message
- comparer les deux éléments

Signatures, application

Alice fera ça:

```
openssl dgst -sha256 -sign alice_private.pem -out message.txt.signature message.txt  
cp message.txt message.txt.signature ../bob
```

Bob fera:

```
openssl dgst -sha256 -verify alice_public.pem -signature message.txt.signature message.txt
```

Vous comprendrez mieux le rôle des signatures dans les token JWT de type JWS: comme la donnée envoyée est simplement encodée en base64, il serait facile de modifier la partie centrale pour s'ajouter des droits par exemple.

Grâce à la signature, on peut rejeter le token si on détecte que sa signature n'est pas valide. Encore faut-il que cette signature soit vérifiée 😊.

Certificats, kezaiko ?

Un certificat est un document qui contient des informations comme:

- une clé publique qui sera à utiliser pour des échanges sécurisés avec le serveur
- un nom de domaine pour lequel la clé servira (possibilité de wildcard)
- des infos sur ce qui détient le serveur et ce qu'est le serveur
- une période de validité (start date - end date)
- une signature venant d'une CA (Certificate Authority) sensée avoir délivré le certificat en ayant validé les informations et attestant de son sang (😈) que c'est bien nous qui controlons le nom de domaine et pas un voisin qui veut se faire passer pour nous.

Certificats, ça sert à quoi ?

C'est donc un moyen de:

- se présenter (bon serveur ?)
- présenter notre clé publique aux personnes ou machines qui souhaiteraient communiquer avec nous de manière sécurisée.

Simple non ? Si on fait confiance à la CA, on peut donc faire confiance aux données que cet organisme a signé.

Certificats, ça ressemble à quoi DER / PEM ?

C'est un fichier PEM comme pour des clés mais avec des entêtes/footers:

- -----BEGIN CERTIFICATE----- au lieu de -----BEGIN PUBLIC KEY-----
- -----END CERTIFICATE----- au lieu de -----END PUBLIC KEY-----

La norme X.509 définit le format de ces certificats.

i Si vous avez un fichier qui n'a pas l'air d'être du texte ressemblant, vous avez peut être directement le certificat sous format DER (le contenu entre les balises, décodé du base64).

Pour convertir de l'un vers l'autre on a:

```
openssl x509 -in certif.der -inform DER -outform PEM -out certif.pem
```

Certificats, ça s'obtient comment ?

Quand on veut un certificat signé par une CA, on remplit une CSR (Certificat Signing Request) qui est aussi au format PEM.

```
openssl req -out CSR.csr -key alice_private.pem -new
```

i Il est aussi possible de générer directement une clé et une CSR en même temps:

```
openssl req -out CSR.csr -new -newkey rsa:2048 -nodes -keyout privateKey.pem
```

Vous envoyez le fichier CSR à la CA, elle fait son enquête et si tout est ok, la CA signera votre CSR et le resultat sera votre certificat.

Certificats, quelques commandes utiles

Sortir quelques informations sur un certificat: les dates de validité et le sujet

```
openssl x509 -in certificat.pem -noout -subject -startdate -enddate
```

Vérifier que le certificat et une clé privée matchent, comparez le modulus contenus dans les deux objets concernés.

```
openssl x509 -noout -modulus -in certificate.crt | openssl md5  
openssl rsa -noout -modulus -in privateKey.key | openssl md5
```

D'autres sont présentes sur le site sslshopper.com

Bref, c'est fini !

Merci à vous

