Quelques utilisations de la cryptographie

Olivier ROUSSEL
olivier.roussel@univ-artois.fr

Notions de cryptographie

Condensé de message, Hachage cryptographique, Empreinte (message digest, cryptographic hash, fingerprint)

- Valeur $h(m)$ calculée à partir d'un message m et permettant de garantir son intégrité. La fonction utilisée doit être telle que
  - Un changement d'un seul bit du message m change complètement le condensé $h(m)$.
  - Il n'est pas possible en pratique de générer un message produisant un condensé choisi à l'avance.
  - La probabilité d'avoir deux messages intelligibles avec le même condensé est faible (collision)
- La taille de $h(m)$ (nombre de bits) est normalement fixe.
- Utilisé pour le chiffrement des mots de passe et pour la vérification des fichiers (ex: sha512sum)
- Exemples d'algorithme : md5 (obsolète), sha1 (vieillissant), sha256, sha512.

Vérification de l'intégrité de fichiers

- `md5sum fichier` (obsolète)
- `sha1sum fichier`
- `sha256sum fichier`
- `sha512sum fichier`

Gestion des mots de passe

- On ne peut pas stocker le mot de passe p d'un utilisateur en clair dans un fichier (si un pirate ou l'administrateur y accède, il obtient tous les mots de passe !).
- L'idée est de stocker $h(p)$ (dans le fichier /etc/shadow sous Unix). Pour vérifier que le mot de passe p' tapé par l'utilisateur est le bon, on calcule $h(p')$ et on compare à $h(p)$.
- Le pirate peut néanmoins tenter une attaque par rainbow table. Il crée une liste de mots de passe p possibles, précalcule $h(p)$ et stocke $p + h(p)$ dans un fichier. S'il met la main sur $h(p)$, il peut faire une recherche dans sa table.
- Pour rendre cette attaque impraticable, on ajoute une chaîne de salage (salt) s choisie aléatoirement, et on stocke $s + h(p + s)$.
- Pour vérifier que le mot de passe p' tapé par l'utilisateur est le bon, on récupère s dans le fichier, on calcule $h(p' + s)$ et on compare à $h(p + s)$.

Cryptographie symétrique (à clef secrète)

- L'algorithme de chiffrement f est connu de tous. On utilise une clef k qui permet de rendre le message m incompréhensible. Le message chiffré est $f(k, m)$.
- La même clef est utilisée pour chiffrer et déchiffrer les données (chiffrement symétrique). Donc $f^{-1}(k, f(k, m)) = m$.
- La clef doit donc rester secrète (difficile)
- Exemples d'algorithmes : DES (obsolète), triple-DES, RC2, RC4, Skipjack, IDEA, Blowfish, AES...

Cryptographie asymétrique (à clef publique)

- Le chiffrement est asymétrique. On n'utilise pas la même clef pour chiffrer et déchiffrer.
- Une clef k_{pub} est publique et peut être distribuée largement, l'autre k_{priv} est privée et doit rester secrète.
- Selon le but que l'on veut atteindre, on chiffre avec l'une ou l'autre clef et on déchiffre avec la seconde clef. Donc, $f^{-1}(k_{pub}, f(k_{priv}, m)) = f^{-1}(k_{priv}, f(k_{pub}, m)) = m$.
- Pour envoyer un message secret à une personne, on le chiffre avec la clef publique de cette personne. Seul celui qui a la clef privée correspondante pourra déchiffrer le message.
- Exemple : RSA basé sur l'exponentiation modulaire et la factorisation des nombres

Considérations pratiques

- Les algorithmes à clef secrète sont rapides (on sait chiffrer en temps réel)
- Les algorithmes à clef publique sont plus lents

Signature électronique

- On intègre son identité au message. On calcule alors un condensé (plus petit à chiffrer) que l'on chiffre avec sa clef privée. Donc, on calcule $f(k_{priv}, h(m + id))$.
- Le correspondant peut vérifier notre identité et l'intégrité du message en déchiffrant avec notre clef publique

Man In the Middle Attack

- Un pirate peut s'interposer au milieu d'une communication, intercepter les messages et en remplacer certains (comme un proxy).
- Exemple : Bob demande à Alice sa clef publique, mais c'est Charlie qui intercepte le message de Bob et lui retourne sa propre clef publique (celle de Charlie) en la faisant passer pour celle d'Alice. Bob croit que seule Alice pourra déchiffrer ses messages, alors qu'en réalité seul Charlie peut le faire. Charlie peut chiffrer les message interceptés avec la clef publique d'Alice et les lui transmettre, pour se rendre quasiment invisible.

Certificat

- Pour être sûr qu'une clef que l'on reçoit est bien celle de la personne avec qui l'on communique, on fait intervenir un tiers de confiance T qui vérifie l'identité et signe la clef de cette personne. La clef publique du tiers (largement diffusée) permet de vérifier le certificat.
- Le certificat d'une personne A contient donc $k^A_{pub} + id_A + f(k^T_{priv}, h(k^A_{pub} + id_A))$
- Il existe divers tiers de confiance (Certificate Authorities (CA)): Verisign, GTE, IBM,...

Vérification de l'identité d'une personne

- On peut vérifier l'identité d'une personne avec qui on dialogue en utilisant sa clef publique.
- Il suffit de choisir une information aléatoire r, que l'on chiffre avec la clef publique de la personne.
- On transmet alors $f(k_{pub}, r)$ à notre correspondant en lui demandant de le déchiffrer.
- Seul le possesseur de la clef privée correspondant à la clef publique utilisée est en mesure de nous transmettre r.

SSH

SSH avec mot de passe

- Pour se connecter à distance sur une machine
  `ssh remoteUser@remoteHost`
- Pour exécuter une commande à distance sur une machine
  `ssh remoteUser@remoteHost commande`
- Il faut prouver à la machine remoteHost qu'on est bien l'utilisateur remoteUser : par défaut, cela se fait en tapant un mot de passe.

Identité de la machine

- Quand on se connecte pour la première fois à une machine, rien ne garantit qu'un pirate n'intercepte pas notre connexion. On nous affiche donc l'empreinte de la clef publique de la machine pour vérifier son identité et on nous demande de confirmer que c'est la bonne machine.
- Il faut donc obtenir par un moyen sécurisé l'empreinte de la clef de la machine sur laquelle on se connecte. Cette empreinte s'obtient en tapant sur la machine elle même `ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key.pub` ou `ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub` selon le type de clef présentée.

Identité de la machine (2)

- Une fois l'empreinte vérifiée, elle est enregistrée dans le fichier `~/.ssh/known_hosts` ce qui permet de faire la vérification automatiquement pour les futures connexions.
- Quand l'empreinte de la machine change (suite à une réinstallation ou à une attaque de type man in the middle), la connexion est annulée. Il faut vérifier la raison de ce changement et le cas échéant, supprimer l'ancienne empreinte du fichier `~/.ssh/known_hosts` pour pouvoir enregistrer la nouvelle.

SSH sans mot de passe

- Pour ne plus avoir à taper son mot de passe, il faut générer sur localHost un couple clef privée/clef publique :
  `ssh-keygen`
- Cela génère dans `~/.ssh` un fichier `id_rsa` (clef privée) et un fichier `id_rsa.pub` (clef publique). La clef privée doit être protégée et ne jamais sortir de la machine !
- Pour protéger la clef privée, on peut :
  - soit choisir des droits d'accès au fichier `id_rsa` qui garantissent que personne d'autre ne pourra accéder à ce fichier (attention aux partages réseaux !)
  - soit protégér cette clef privée par un mot de passe (mais alors il faudra taper ce mot de passe à chaque fois qu'on a besoin de la clef, ou utiliser ssh-agent pour le fournir à notre place).

SSH sans mot de passe (2)

- Pour autoriser un utilisateur X à se connecter au compte `remoteUser@remoteHost` sans fournir de mot de passe, il suffit d'ajouter sa clef publique au fichier
  `remoteHost:~remoteUser/.ssh/authorized_keys`
- Cela peut se faire
  - à la main : `localHost> cat ~/.ssh/id_rsa.pub | ssh remoteUser@remoteHost 'cat >> ~/.ssh/authorized_keys'`
  - automatiquement : `ssh-copy-id remoteUser@remoteHost`

ssh-agent

- Quand on choisit de protéger la clef privée par un mot de passe, on peut demander à `ssh-agent` de conserver la clef privée et d'effectuer les calculs basés sur cette clef pour le compte des clients ssh (la clef n'est pas transmise, mais cela prouve que l'on dispose de la clef privée).
- Cela permet de ne taper le mot de passe qui protège la clef qu'une seule fois, quel que soit le nombre d'utilisations de ssh.

ssh-agent (2)

- La variable d'environnement `SSH_AUTH_SOCK` indique au client ssh comment contacter ssh-agent (via une socket unix).
- `ssh-agent` *commande* lance à la fois `ssh-agent` et la commande (comme processus fils) et définit les variables d'environnement automatiquement. Normalement, ssh-agent est lancé automatiquement dès la connexion à la machine locale.
- Si nécessaire, pour enregistrer une clef privée dans `ssh-agent`, on utilise `ssh-add` (sans option pour enregistrer la clef par défaut). La variable `SSH_ASKPASS` indique à `ssh-add` comment demander le mot de passe.

GPG

Génération d'un couple de clefs

- `gpg2 --gen-key`
- Il faut choisir
  - le type de clef
  - la taille de la clef (une clef longue est plus sûre, mais nécessite plus de calculs)
  - la durée de validité de la clef
  - le nom
  - l'adresse email
  - un éventuel commentaire ou surnom
  - une phrase de passe pour protéger l'accès à la clef privée
- La génération de la clef nécessite l'utilisation de nombres vraiment aléatoires (obtenus à partir des frappes au clavier, déplacements de souris, etc.)
- la clef privée et la clef publique sont ajoutées à notre trousseau (`~/.gnupg/secring.gpg` et `~/.gnupg/pubring.gpg`)

Signature d'un fichier

- `gpg2 --clearsign fichier`
- `fichier.asc` contient le fichier suivi de la signature du fichier

Vérification de la signature d'un fichier

- la clef publique du signataire doit être dans notre trousseau
- `gpg2 --verify fichier.asc`

Exporter une clef publique

- `gpg2 --armor --output pub.key --export 'Jean DUPONT'`
- Cette clef publique peut être envoyée par mail, placée sur le web ou dans un entrepôt de clef gpg.

Importation d'une clef dans le trousseau

- `gpg2 --import pub.key`

Chiffrement symétrique

- `gpg2 -c fichier`
- Génère un fichier `fichier.gpg`
- Nécessite la saisie de la clef de chiffrement (double saisie)

Chiffrement asymétrique

- `gpg2 -e -r 'Jean DUPONT' fichier`
- On doit donner l'identité du destinataire (recipient) et sa clef publique doit être dans notre trousseau
- On peut ajouter l'option `--sign` pour en plus signer le fichier

Déchiffrement symétrique ou asymétrique

- `gpg2 -d fichier.gpg`
- Par défaut, le résultat est affiché à l'écran. Utiliser `--output` pour sauver le résultat dans un fichier.

Commandes diverses

- `gpg2 --list-keys`
  afficher les clefs publiques que l'on possède (voir aussi `--list-secret-keys`)
- `gpg2 --delete-key 'Propriétaire de la clef'`
  supprimer une clef publique importée
- `gpg2 --delete-secret-and-public-key 'Propriétaire de la clef'`
  supprimer notre clef privée et la clef publique correspondante (en cas de doublon)

PKI

PKI

- Une PKI (Public Key Infrastructure, infrastructure de gestion des clefs) est un système qui permet de créer et gérer des certificats. Ce système est utilisé par le tiers de confiance.
- La PKI permet de
  - générer/renouveller/publier des certificats
  - révoquer des certificats
  - éventuellement archiver les clefs qui servent uniquement au chiffrement

Fonctionnement global de la PKI

- La PKI dispose de son certificat (soit autosigné, soit signé par une autre autorité de certification (CA)) et de sa clef privée (CA.crt et CA.key)
- Un utilisateur transmet à la PKI une Certificate Signing Request (CSR) qui contient son identité et sa clef publique.
- Après vérification, la PKI signe le certificat et le retourne à l'utilisateur. Quand le certificat doit servir à l'authentification, la clef privée de l'utilisateur ne doit jamais être connu de la PKI (non répudiation).
- La PKI publie une liste de certificats qui ne sont plus valides (certificats révoqués)

Exemples de PKI

- Il existe différentes PKI open-source :
  - dogtag-pki
  - openCA
  - EJBCA
  - ...
- easy-rsa est une PKI miniature et simplifiée à l'extrême. C'est un ensemble de scripts initialement distribués dans openvpn qui simplifient l'utilisation de la commande openssl (c'est elle qui fait le travail).

easy-rsa (initialisation de la CA)

Sur la machine de la CA, l'administrateur tape

```
# copie des scripts
cp -r /usr/share/easy-rsa/3/ easy-rsa
cd easy-rsa
./easyrsa init-pki
./easyrsa build-ca
```

On peut fournir plus d'informations sur le propriétaire du certificat en éditant le fichier de configuration `vars` comme suit (à faire avant l'initialisation)

```
cp /usr/share/doc/easy-rsa/vars.example vars
set_var EASYRSA_DN "org"
set_var EASYRSA_REQ_COUNTRY "FR"
set_var EASYRSA_REQ_PROVINCE "Hauts-de-France"
set_var EASYRSA_REQ_CITY "Lens"
set_var EASYRSA_REQ_ORG "LaBoite SA"
set_var EASYRSA_REQ_EMAIL "me@laboite.fr"
set_var EASYRSA_REQ_OU "Département informatique"
```


Détail des champs demandés

- Country Name : 2 lettre en majuscules (FR)
- State or Province Name (full name) : l'état dans le pays (ex: Texas)
- Locality Name (eg, city) : la ville
- Organization Name (eg, company) : le nom de l'entreprise
- Organizational Unit Name (eg, section) : le nom du département dans l'entreprise
- Common Name (eg, your name or your server's hostname) : le nom de la personne ou du serveur
- Email Address

easy-rsa (demande de certificat (CSR))

Sur son poste, Alice lance

```
cp -r /usr/share/easy-rsa/3/ easy-rsa
cd easy-rsa
./easyrsa init-pki
./easyrsa gen-req alice
```

Elle envoie `pki/reqs/alice.req` à la CA. Sa clef privée doit rester sur sa machine (on la trouve dans `pki/private/alice.key`).

easy-rsa (signature de certificat)

La CA signe le certificat en faisant :

```
# import du certificat et assignation d'un nom court
./easyrsa import-req /chemin/alice.req alice

# signature d'un certificat pour un client
#                 (utilisateur ou machine cliente)
./easyrsa sign-req client alice

# ou bien pour un serveur
#./easyrsa sign-req server LeNomCourtDuCertificat
```

On peut visualiser le certificat ou la requête avec

```
./easyrsa show-cert alice
./easyrsa show-req alice
```

La CA envoie `pki/issued/alice.crt` à Alice.

easy-rsa (récupération du certificat)

Alice récupère le certificat et le range dans `pki/issued/alice.crt`.
Alice va alors générer un fichier au format PKCS#12 pour importation dans Firefox. Deux solutions :

- `./easyrsa export-p12 alice noca`
  `# le fichier généré est dans pki/private/alice.p12`
- `openssl pkcs12 -export -name "Certificat d'Alice"`
  `-inkey pki/private/alice.key`
  `-in pki/issued/alice.crt -out alice.p12`
  Le nom sert à identifier facilement le certificat.

Le mot de passe d'exportation demandé sert à protéger la clef dans le fichier .p12

easy-rsa (révocation d'un certificat)

L'administrateur de la CA peut révoquer un certificat en tapant :

```
./easyrsa revoke alice
# génération de la CRL (Certificate Revocation List)
./easyrsa gen-crl
```

Le fichier `pki/crl.pem` liste les certificats qui ne sont plus valides. Il doit être récupéré par tous les systèmes qui vérifient les certificats pour qu'ils puissent rejeter les certificats révoqués.

Authentification apache

Utilisation d'un certificat

- On peut exiger que le navigateur s'authentifie via un couple clef privée/clef publique avant d'accéder à une ressource web.
- Le certificat et la clef privée seront enregistrés dans le navigateur.
- Le navigateur devra fournir le certificat du client au serveur web
- On doit enregistrer sur le serveur le certificat de l'autorité qui a signé le certificat du client.
- Le serveur web vérifiera la signature du certificat. Si le certificat est signé par la bonne autorité, il laissera le client accéder. Le serveur envoie une donnée chiffrée avec la clef publique du client pour vérifier que le navigateur possède bien la clef privée correspondante (ce qui authentifie l'utilisateur).

Configuration du serveur

Dans /etc/httpd/conf.d/ssl.conf

```
# par défaut, on n'exige pas de certificat du client
SSLVerifyClient none

# le certificat de l'autorité qui signe les
# certificats des utilisateurs autorisés
SSLCACertificateFile /etc/pki/tls/certs/ca.crt

# gérer les révocations
#SSLCARevocationCheck chain
#SSLCARevocationFile crl.pem
```

Dans un fichier de configuration quelconque

```
# le répertoire à accès protégé
<Location /secure/area>
  # accès uniquement par https
  SSLRequireSSL
  # on exige la présentation d'un certificat
  SSLVerifyClient require
  # ici, on choisit de n'accepter que les certificats
  # signés directement par la CA
  SSLVerifyDepth 1
</Location>
```


Dans le navigateur

- Il faut importer votre certificat et votre clef privée.
- Firefox accepte des imports au format PKCS#12 : Edit/Preferences/Advanced/Encryption/Your Certificates/Import

SSLVerifyClient global/local

`SSLVerifyClient require` peut être spécifié

- pour tout le site
  Dans ce cas, le certificat client est envoyé au début de la communication TLS, et il n'y a pas de problème.
- dans une directive <Directory> ou <Location>
  Dans ce cas, le client envoie la requête HTTP sans transmettre le certificat client. Quand le serveur voit qu'un certificat est requis, il renégocie la communication TLS pour demander au client d'envoyer un certificat (ou demande une Post Handshake Authentication en TLSv1.3). Cela est incompatible avec HTTP/2. De ce fait, avec TLSv1.3, le navigateur n'envoie pas le certificat. Dans firefox, on peut activer l'envoi dans about:config en mettant à true la variable security.tls.enable_post_handshake_auth. Une autre possibilité serait de désactiver TLSv1.3 dans Apache avec la directive SSLProtocol all -SSLv3 -TLSv1.3. Mais ce n'est bien sûr pas une option en pratique.