Notions de cryptographie

Table des matières

1 Notions de base

- Le chiffrement : consiste à passer d'un texte en clair à un texte chiffré en utilisant une clé de chiffrement.
- Le déchiffrement : consiste à passer d'un texte chiffré à un texte en clair en utilisant une clé de déchiffrement.
- Le décryptage : consiste à passer d'un texte chiffré à un texte en clair **sans** utilisation de clé.

- La cryptographie : est la création d'algorithmes de protection des messages.
- La cryptanalyse : est la création de méthodes de déprotection des messages.
- La cryptologie : est l'ensemble de la cryptographie et de la cryptanalyse.

1.1 Services et mécanismes

La cryptographie fournit des services :

- Confidentialité : le message ne peut pas être lu
- Intégrité : le message ne peut pas être modifié
- Authentification : s'assurer de l'identité de quelqu'un
- Non-répudiation : empêcher quelqu'un de dire qu'il n'est pas l'auteur d'un message

Ces services sont fournis grâce à des mécanismes :

- Chiffrement
- Scellement et signature

1.2 Algorithmes de chiffrement

Deux types de chiffrement existent suivant le type de clés utilisées.
Dans les chiffrements symétriques **à clé secrète**, la même clé sert à chiffrer et déchiffrer. Ces algorithmes sont les plus rapides. Ils sont utilisés principalement dans le chiffrement de données.
Dans les chiffrement asymétriques **à clé publique**, la clé publique permet de chiffrer et la clé privée permet de déchiffrer. Ces algorithmes sont les plus lents. Ils sont utilisés principalement dans l'authentification et la signature.

1.2.1 Algorithmes symétriques

Deux types d'algorithmes symétriques existent suivant le type des données chiffrées.
Les chiffrements de flux permettent de chiffrer un flux de données bit par bit. Un exemple est le chiffrement **RC4**.
Les chiffrements de blocs permettent de chiffrer les données blocs par bloc. La taille des clés et la taille des blocs varient d'un algorithme à l'autre. Les principaux exemples sont :

- **DES** (clé 56 bits, bloc 64 bits), 3DES (2/3 clés)
- **AES** (rijndael) (clés 128,192 et 256 / bloc 128)
- **Blowfish** (clé 32-448 / bloc 64), **Twofish** (clés 128,192 et 256 / bloc 128)
- **IDEA** (clé 128, bloc 64), **CAST-128** (clé 40-128, bloc 64) **CAST-256** (clé 128-256, bloc 128), **Camellia** (clé 128, 192, 256 / bloc 128)

1.2.2 Algorithmes asymétriques

Dans les algorithmes asymétriques, les clés sont utilisées différemment. La clé publique est générée à partir de la clé secrète. La connaissance de la clé publique ne permet pas la déduction de la clé secrète.
Les principaux algorithmes asymétriques sont :

- Diffie & Hellman
- RSA

Ces algorithmes sont basés sur des problèmes complexes tels que les logarithmes discrets ou la factorisation de grands nombres.
Ces algorithmes sont lents et utilisés principalement pour :

- L'échange de clés symétriques (de session) :
  - l'expéditeur chiffre la clé de session avec la clé publique du destinataire.
  - le destinataire déchiffre avec sa clé privée pour obtenir la clé de session.
- La signature : le haché d'un document est chiffré avec une clé privée.

1.3 Fonctions de hachage

Les fonctions de hachage permettent de générer une empreinte ou un haché, par exemples d'un document, d'un flux de données…
Le haché a toujours la même taille.
Une fonction de hachage doit répondre à ces deux critères :

- il est difficile d'inverser un haché, ie de trouver un texte dont le haché est celui donné
- il est difficile de trouver deux données avec le même haché

Les principaux algorithmes sont :

- MD5 : haché de 128 bits
- SHA-1 (SHA-0) : haché de 160 bits
- SHA-2 : hachés de 224, 256, 384 et 512 bits
- SHA-3 : hachés de 224, 256, 384 et 512 bits.

MD5 n'est quasiment plus utilisé et **SHA-1** est en cours de remplacement par **SHA-2**.
SHA-3 a été conçu suite à la découverte de faiblesses dans les algorithmes précédents et fonctionne donc sur un mode totalement différent. Il est tout de même à noter que les attaques contre **SHA-1** n'ont pas pu être transcrites sur **SHA-2** à l'heure actuelle.

1.4 Algorithmes d'échange de clé

Les algorithmes d'échange de clé permettent à deux personnes, avec chacune un couple de clés asymétriques, de s'échanger une clé symétrique nommée clé de session :

- La clé de session (la clé secrète d'un algorithme symétrique) est chiffrée avec la clé publique du destinataire
- Le destinataire déchiffre avec sa clé privée pour obtenir la clé secrète de session
- Dès que l'expéditeur voit que le destinataire a pu utiliser la clé secrète il a authentifié le destinataire par la même occasion car il a prouvé qu'il connaît la clé privée associée à la clé publique utilisée.

Sur le même schéma, l'authentification peut être réalisée par l'échange d'un aléa chiffré avec la clé publique du destinataire. Le destinataire doit renvoyer l'aléa en clair à l'expéditeur.

1.5 Clés publiques et certificats

L'identification de clés publiques est réalisée par des certificats émis par des autorités de certification reconnues.
L'autorité signe avec sa propre clé privée la clé publique d'une personne ou d'un serveur, ce qui génère un certificat.
Cette personne s'identifie en présentant son certificat.
Une autre personne vérifie l'identité de cette personne en vérifiant la signature du certificat, pour cela elle doit posséder la clé publique de l'autorité reconnue. Il est alors possible d'authentifier cette personne avec la clé publique contenue dans le certificat.
Les clés sont organisées hiérarchiquement, il existe des autorités intermédiaires dont les clés publiques n'ont pas besoin d'être connues.
Les clés privées sont stockées de façon sécurisée, notamment celles des autorités de certification.
Les navigateurs et/ou systèmes d'exploitation contiennent des certificats d'autorités connues, mais pas forcément reconnues par leurs utilisateurs, cela dépendant de la politique de l'autorité.
Un certificat contient plusieurs données :

- l'identité du propriétaire,
- les dates de validité du certificat et
- les usages de la clé privée correspondante.

Il contient également l'URL de la **CRL** qui gère les révocations de certificats avant leurs dates de fin de validité, par exemple en cas de vol d'une clé privée.

1.6 PFS (Perfect Forward Secrecy)

La **PFS** est une propriété garantissant la confidentialité du chiffré, même en cas de vol de la clé privée d'un serveur.
SSH possède la **PFS** par conception.
SSL ne la possède que lors de l'utilisation d'algorithmes d'échange de clé le permettant comme les algorithmes d'export (40 bits – donc à ne jamais utiliser) et les algorithme EDH (Diffie Hellman Éphémère).

2 Algorithmes de chiffrement réseaux

2.1 SSH

SSH est un remplacement sécurisé de **rsh**, **rlogin** et **rcp** avec de nombreux avantages. Le premier est l'authentification forte (asymétrique) : le serveur présente sa clé publique, le client un mot de passe ou une clé privée. Le deuxième avantage est le chiffrement fort (symétrique). Enfin **SSH** implémente la « Perfect Forward Secrecy ».
SSH permet le multiplexage de différents types de canaux : accès à un shell interactif, copie de fichiers, transfert de ports TCP, de session X11, etc.
Deux protocoles coexistent : v1.5 et v2.0 (RFC 4251-6), mais ils ne sont pas inter-opérables.
SSH permet de « sécuriser » le transfert, mais peut être utilisé de façon vulnérable, par exemple avec une analyse passive. Par exemples : nombre de caractères par commande, caractères avec/sans écho, i.e. nombre de caractères par mot de passe, vitesse de frappe, etc.
Une autre utilisation vulnérable est l'**attaque de l'intercepteur** : l'utilisateur doit accepter une nouvelle clé.
Références :

- http://www.openwall.com/advisories/OW-003-ssh-traffic-analysis/
- http://naughty.monkey.org/~dugsong/dsniff/
- http://www.hsc.fr/ressources/presentations/mitm/

2.1.1 OpenSSH

- http://www.openssh.com/

OpenSSH est une implémentation libre (BSD 2 clauses) des protocoles 1.3, 1.5 et 2.0.
Développé sous **OpenBSD** puis porté sur tout autre **Unix**.
Le code est développé de façon sécurisée, avec de la séparation des privilèges pour combattre les failles dans les bibliothèques tierces.
Utilise **OpenSSL** comme moteur de chiffrement.
Limitation des utilisateurs :

```
AllowGroups  AllowUsers  DenyGroups  DenyUsers
PermitRootLogin (without-password, forcedcommands-only...)
```

Limitation des commandes :

```
ForceCommand
```

Limitation des authentifications :

```
PasswordAuthentication  PermitEmptyPasswords  PubkeyAuthentication
```

Limitation des tunnels :

```
AllowTcpForwarding  PermitOpen  PermitTunnel  X11Forwarding
```

Limitation des privilèges du démon :

```
UsePrivilegeSeparation
```

Possibilité d'authentification forte avec génération des clés sur le client par ssh-keygen, puis copie de la clé publique sur le serveur dans **~/.ssh/authorized_keys**
Limitation de l'utilisation des clés publiques :

```
command=        from=
no-port-forwarding=   no-X11-forwarding=
permitopen=       tunnel=
```

D'autres implémentations existent comme **mindterm** (client applet java), **putty**/**filezilla** (divers clients Windows), **corkscrew**/**proxytunnel**, **SSHFS** (FS sur **SFTP**) et **ssh.com**/**dropbear** (clients/serveurs)

2.1.2 rsh, rlogin, rcp et telnet

Ces protocoles ont de nombreuses failles comme l'authentification et les échanges en clair et non protection contre la modification ou l'insertion.
La notion de « site sûr » (r-commandes) est intrinsèquement **vulnérable** car elle permet à une intrusion de se répandre.
SSH permet de remplacer les commandes **rsh**, **rlogin** et **rcp** en évitant ces failles. **SecSH** possède **sftp** pour remplacer la commande **ftp**.

2.2 SSL et TLS

SSL v3.0 (Secure Sockets Layer) a été développé par **Netscape** http://wp.netscape.com/eng/ssl3/
TLS v1.0 (Transport Layer Security) a été développé par l'**IETF** (RFC 2246) à partir de **SSLv3**.
Ils sont utilisés en encapsulation (ex. **HTTP** - RFC 2818) ou en upgrade (ex. **SMTP** - RFC 2487).
Ils comportent de nombreuses fonctions optionnelles :

- Chiffrement fort (symétrique)
- Authentification forte (asymétrique) avec certificats

L'authentification requière la confiance dans les autorités de certification et dans les utilisateurs.
Deux outils permettent de déchiffrer les flux :

- http://monkey.org/~dugsong/dsniff/
- http://www.rtfm.com/ssldump/

Pour se protéger, il faut utiliser les protocoles d'échange de clés avec la Perfect Forward Secrecy, comme les algorithmes « export » ou basés sur Ephemeral Diffie-Hellman.

2.2.1 Utilisation avec SMTP et HTTP

Dans **SMTP Over TLS** (RFC 2487), **TLS** est utilisé en upgrade. **TLS** permet l'authentification des sites de confiance et le relayage des itinérants. Le chiffrement ne se fait qu'entre 2 serveurs et permet donc toujours le filtrage de contenu.
Dans **HTTP Over TLS** (RFC 2818), **TLS** est utilisé en encapsulation. Le serveur doit s'authentifier.

2.2.2 OpenSSL / GNUTLS

OpenSSL http://www.openssl.org/ est l'implémentation de référence **SSL v2/v3** & **TLS v1.x**.
OpenBSD a procédé à un fork nommé **LibreSSL** https://www.libressl.org/ pour améliorer la sécurité, le code de base et appliquer les meilleures pratiques.
GNUTLS http://www.gnutls.org/ est une implémentation alternative de **TLS 1.2** & **SSL 3.0**.
Ils supportent l'authentification et le chiffrement forts, le **DTLS** (Datagram), les courbes elliptiques, les cartes accélératrices et les jetons d'authentification (via des modules externes).

2.2.3 Généralisation de TLS/SSL

stunnel http://stunnel.mirt.net/ permet de rajouter le support de **SSL/TLS** aussi bien des côtés clients que serveurs.
Côté serveur **stunnel** permet d'écouter sur un port chiffré et de renvoyer le flux déchiffré sur un autre port.
Côté client **stunnel** permet d'écouter sur un port en clair et de renvoyer le flux chiffré sur un autre port.
stunnel permet d'effectuer de l'authentification par les certificats.

2.2.4 VPN SSL

Les **VPN SSL** encapsulent **IP** dans **SSL**, souvent par l'intermédiaire de **PPP**.
Attention, les **VPN SSL** ne sont pas compatibles entre eux, ni avec **IPSec**. Les fonctions d'authentification dépendent des implémentations.

Penser à mettre en place du filtrage en sortie de tunnel : avoir un accès "**sécurisé**" à un réseau ne doit pas forcément donner accès à tout ce réseau.
OpenVPN http://openvpn.net/
SSLTunnel http://www.hsc.fr/ressources/outils/ssltunnel/

2.3 Chiffrement de fichiers et de mails

2.3.1 Chiffrement de fichiers

GnuPG http://www.gnupg.org/
Remplacement OpenSource de **PGP**, il implémente le protocole **OpenPGP** : chiffrement, signature, gestion de clés… Il supporte **S/MIME** depuis la v2

2.3.2 Chiffrement de mails

Il existe 2 protocoles incompatibles : **OpenPGP** et **S/MIME**. Tous les clients de messagerie ne sont pas compatibles avec les deux protocoles. Les clés et leurs identifications sont incompatibles.
OpenPGP nécessite des clés **PGP**, génère un message au format **OpenPGP** qui contient le chiffré et/ou la signature
S/MIME nécessite des certificats **X509**, génère un message au format **S/MIME** qui contient le chiffré et/ou la signature

3 Tests de chiffrement SSL/TLS

Openssl http://www.openssl.org/
Full-strength general purpose cryptography library (including **SSL** and **TLS**)

```
$ openssl list-cipher-algorithms
DES-EDE3-CBC
```

SSLScan https://github.com/ioerror/sslscan
SSLScan determines what ciphers are supported on **SSL**-based services.

```
$ sslscan --no-failed --ssl3 10.5.5.6:443
Accepted SSLv3 168 bits DES-CBC3-SHA
```

Connexion :

```
$ openssl s_client -connect 10.5.5.6:443 -cipher DES-CBC3-SHA
```

4 Attaques MITM

4.1 Principe

Man In The Middle (**MITM**) = Attaques de l'intercepteur
Permet de déchiffrer un flux chiffré en étant entre les protagonistes.
Le client se connecte à l'attaquant qui se connecte au serveur.
Le client chiffre avec l'attaquant qui chiffre avec le serveur.
L'attaque échoue si le client vérifie l'authentification du serveur ou vice-versa, ce qui ne peut être réalisé que

par authentification forte (bi-clé).
En pratique, contre **HTTPS** ça marche toujours : l'utilisateur ignore l'avertissement du navigateur. Contre **SSH** ça peut marcher lors de la première connexion car l'utilisateur n'appelle pas l'administrateur pour avoir confirmation de la clé publique du serveur.

4.2 MITM vs SSH

L'attaquant peut voir les login/mdp du client, en clair, voir l'écran du client et déconnecter le client pour prendre le contrôle de la connexion, après qu'il ait obtenu des privilèges d'administration.
L'attaquant ne peut pas présenter la même clé publique que le serveur, ni authentifier le client avec sa clé publique.

4.2.1 sshmitm

Dans le fichier **/etc/ssh/sshd_config** s'assurer d'avoir :

```
Protocol 1,2
PasswordAuthentication yes
HostKey /etc/ssh/ssh_host_key
```

Lancer « ssh-keygen -A » si les fichiers **ssh_host_rsa_key*** n'existent pas dans **/etc/ssh**
Relancer le serveur ssh : **/etc/init.d/ssh restart**
Dans une première console, lancer **sshmitm** :

```
$ sshmitm -I -p 2222 127.0.0.1 22
```

Dans une seconde console, lancer **ssh** (il faut désactiver la compression et activer **sshv1**)

```
$ ssh 127.0.0.1 -p 2222 -1 -o Compression=no
```

4.3 MITM vs SSL et TLS

4.3.1 ssldump

ssldump déchiffre le trafic **SSL** enregistré dans des fichiers **libpcap** ou capturé.
Il nécessite la clé privée du serveur.
Il peut par exemple déchiffrer du traffic **HTTPS**.
Il faut mettre en place un relais transparent, par exemple **squid** en mode relais transparent.
Cela ne fonctionne que si le client utilise l'entête **Host:** .
Limiter les algorithmes à ceux qui ne supportent pas la **PFS**.
Dans le navigateur il faut sûrement désactiver la compression : cf **network.http.acceptencoding** dans **about:config**

4.3.2 Squid en accélérateur

Dans **/etc/squid/squid.conf** s'assurer d'avoir :

```
https_port 3129 key=/etc/ssl/apache2/server.key cert=/etc/ssl/apache2/server.crt
cipher=DES-CBC3-SHA sslflags=NO_SESSION_REUSE options=NO_TLSv1 accel vhost
defaultsite=10.5.5.6 vport=443 protocol=https
#ssl_bump allow all
ssl_bump client-first all
sslproxy_flags DONT_VERIFY_PEER
sslproxy_cert_error allow all
hierarchy_stoplist cgi-bin ?
request_header_access Allow allow all
request_header_access Accept-Encoding deny all
always_direct allow all
```

4.3.3 ***ssldump***

Dans le terminal #1 :

```
# ssldump -Adn -i lo -k /etc/ssl/apache2/server.key
```

Dans le terminal #2 :

```
$ openssl s_client -connect 127.0.0.1:3129 -cipher
DES-CBC3-SHA -ssl3 -no_comp
GET / HTTP/1.0<Enter>
<Enter>
```

Dans le terminal #1 :

```
1 11 10.9468 (0.0000) C>SV3.0(40) application_data
---------------------------------------------------------------
--
GET / HTTP/1.0
---------------------------------------------------------------
--
```