Table des matières

[Notion de cryptographie 3](#_Toc80118980)

[Sécurité informatique 3](#_Toc80118981)

[Chiffrements 3](#_Toc80118982)

[Hachage. 3](#_Toc80118983)

[Signature électronique. 4](#_Toc80118984)

[Certificat électronique. 4](#_Toc80118985)

[Algorithme de Diffie-Hellman 4](#_Toc80118986)

[Règles générales. 4](#_Toc80118987)

[La sécurité sur Internet. 5](#_Toc80118988)

[SSH 6](#_Toc80118989)

[Établissement d’une communication 6](#_Toc80118990)

[Fichier de conf 6](#_Toc80118991)

[Implémentation par mot de passe. 7](#_Toc80118992)

[Implémentation par paire de clés. 7](#_Toc80118993)

[Message d’erreur a la connexion sur un serveur quelconque : « REMOTE HOST IDENTIFICATION HAS CHANGED » 7](#_Toc80118994)

[Pratique 7](#_Toc80118995)

[Start / Stop / Restart 7](#_Toc80118996)

[Emplacement des trousseaux de clé SSH 7](#_Toc80118997)

[Fichier de configuration du SERVICE SSHD 7](#_Toc80118998)

[Fichier de configuration du SERVEUR SSHD 7](#_Toc80118999)

[Autoriser la création d’un type de clé. SERVEUR SSHD 8](#_Toc80119000)

[Stopper la création d’un type de clé. SERVEUR SSHD 8](#_Toc80119001)

[Régénération des différents trouceaux SSHD : 8](#_Toc80119002)

[Empreinte (Fingerprint) d’une clé. 8](#_Toc80119003)

[Copie de fichier 8](#_Toc80119004)

[Création trousseau de clé user 8](#_Toc80119005)

[Copie clé publique user 🡪 serv. 8](#_Toc80119006)

[Emplacement des clés autorisé 8](#_Toc80119007)

[Fail Connexion après télécharger une clé 8](#_Toc80119008)

[Fichier de stockage des clés connues 8](#_Toc80119009)

[Espionnage de trame 9](#_Toc80119010)

[Utilisé Tshark pour espionner un échange ssh 9](#_Toc80119011)

[Colorisation des flux de conversation sous SSH 9](#_Toc80119012)

[Tunneling SSH 9](#_Toc80119013)

[Différence Local / Remote 9](#_Toc80119014)

[Local = CLIENT à 2 machines 9](#_Toc80119015)

[Remote = SERVEUR à 2 machines 9](#_Toc80119016)

[Local = CLIENT à 3 machines 9](#_Toc80119017)

[Remote = SERVEUR à 3 machines 9](#_Toc80119018)

[Remarque 9](#_Toc80119019)

[Le Natting / Forwarding 10](#_Toc80119020)

[Forwarding 10](#_Toc80119021)

[Natting : 10](#_Toc80119022)

[Commande supplémentaire 11](#_Toc80119023)

[Outils SSH (client) 11](#_Toc80119024)

# Notion de cryptographie

## Sécurité informatique

Beaucoup de maths derrière. La sécurité informatique = sécurité des échanges.

Imaginons que A veut envoyer un massage à B à travers le net. Si aucune sécurité, il y a possibilité d’interception. 4 grands principes de sécurité (ICAN) :

* Intégrité // Si B reçoit le même message sans altération (vérification au préalable).

Alors ok.

* Confidentialité // Imaginons qu’on envoie un mdp de A à B. Si le message circule tel quel sur le réseau, n’importe qui peut l’intercepter et le lire, pas de confidentialité. À l’inverse, si les données sont chiffrées, le pirate aura du mal à en faire quoi que ce soit.
* Authentification // Quand A parlé à B, les deux parties de l’échange doivent être sûr de l’identité de l’autre. Assurer l’identité d’un utilisateur.
* Non-Répudiation // Les 2 parties ne peuvent pas nier que l’échange a eu lieu.

Si ces 4 principes sont bien respectés, alors l’échange est considéré comme idéal et sécurisé. Par exemple, SSH ne fournit que les 3 premiers, tandis que HTTPS donne les 4.

## Chiffrements

Beaucoup de maths derrière encore. 3 types de chiffrement, basés sur des algorithmes mathématiques. Est considéré comme sûr un algo qui n’a pas encore été craqué. Quel que soit le chiffrement, s’il est utilisé seul, c’est un peu trop faible, à utiliser en complément. Ces trois types de chiffrement :

* Pur : Plus utilisé de nos jours, toujours le même chiffrement, sans paramètres de différentiation du chiffrement. Répond au critère C de ICAN seulement.
* Chiffrement symétrique (clés partagées) : Même chose mais un paramètre rentre en compte. Pour chiffrer un message, on utilise une clé. Une clé = Un paramètre. Les clés sont dites partagées ou secrètes. Machine A et B partagent le même algorithme et la même clé. A chiffre son message et B le déchiffre avec le même algorithme mais inversé. Exemple : Chiffrement de César, AES. Ne répond qu’au critère C aussi.
* Chiffrement asymétrique (trousseau de clés) : La machine qui reçoit génère un trousseau de 2 clés, une KPUB, une KPRIV. Les 2 sont complètement différents, pas possible de retrouver l’une à partir de l’autre. Mais elles sont toutes les 2 mathématiquement équivalentes. La clé publique est destinée est à être donnée, afficher, diffuser. La clé privée en revanche doit rester secrète à tout prix. Si A veut envoyer un message à B, il utilise la clé publique de B. Son message va être chiffré asymétriquement, et B sera le seul à pouvoir déchiffrer, vu qu’il a la clé privée correspondante. Exemple d’algorithme : RSA, ECDSA. Répond aux critères C et A.

## Hachage.

Le *hachage* permet de cocher l’objectif d’intégrité. Le principe est le suivant. On veut envoyer un message et vérifier qu’il n’a pas été modifié à la fin. On va utiliser du hachage. Le message est « passé à la moulinette » grâce à un algorithme de hachage. Il va rester une empreinte ou un hash ou un condensat de taille fixe (128, 256 bits, …) qui permet d’identifier les données de manière unique. Exemple : MD5, quelques empreintes sont équivalentes, donc plus usité sur Linux. SHA, algorithme utilisé majoritairement aujourd’hui. Le hachage permet de cocher le principe d’intégrité. Le hachage seul sert à rien, si pirate au milieu, qu’il modifie le message, si l’empreinte était indiqué directement dans le message, il peut modifier comme il le veut. On envoie sur le réseau qu’un message et pas l’empreinte associée. L’empreinte peut en revanche être affiché publiquement sans problème.

## Signature électronique.

Combinaison de hachage et de chiffrement asymétrique. A veut envoyer un message à B. Ce message passe au hachage. On a donc une empreinte. Il passe cette empreinte par un algo de chiffrement asymétrique avec la KPRIV envoyeur. Est créé ainsi la signature électronique. A la réception, B a le message et sa signature. Il recalcule l’empreinte du message reçu ainsi que la signature chiffrée qui lui donnera l’empreinte d’origine. Il compare les deux, et si OK, l’échange a rempli les objectifs d’intégrité et d’authenticité. Si MITM, il ne sait pas faire grand-chose. Il peut prendre le message et le hacher mais quand il le chiffrera avec sa clé privée, il sera niqué parce que B ne possède pas sa clé publique correspondante pour récupérer la bonne empreinte de base. Il sait qu’il y a eu un problème.

## Certificat électronique.

A pour but de cocher le dernier objectif, la Non-Répudiation. Pour internet et sécurité, SSH parfois pas suffisant. Le top c’est HTTPS car le certificat est présent, les 4 objectifs sont remplis. Un certificat est un fichier (x.509 – rfc5280) qui associe la clé publique d’une personne à son identité. Objectif rempli car vérification de l’authenticité de la clé publique systématique.

Algorithme de Diffie-Hellman.

Alice et Bob veulent partager un secret, et se parler en utilisant une clé symétrique. Ils vont donc se mettre d’accord sur une clé de session. Alice et Bob, chacun de leur côté, choisissent un nombre aléatoire. Ensuite, ensemble, ils déterminent deux nombres. Ces nombres peuvent circuler en clair. Chacun calcul de son côté A = ga mod p pour Alice et B = gb mod p pour Bob. Alice envoie A et Bob envoie B. Au final les deux font le même calcul gab mod p.

Clé = gab mod p.

## Règles générales.

* Plus une clé est courte, plus la vitesse de chiffrement est élevée. Et la réciproque est vrai.
* Quand on *chiffre*, TOUJOURS avec *KPUB* du *destinataire*. Destinataire déchiffre avec sa KPRIV.
* Pour *signer*, l’expéditeur signe avec sa *propre KPRIV*. Le destinataire déchiffre avec la KPUB correspondante.
* Possibilité de chiffrer ET signer un échange réseau. Ce genre d’échange nécessite 3 clés de chaque côté. Les KPUB des 2 et sa propre KPRIV.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Symétrique | Asymétrique | Hachage | Signature | Certificat |
| Intégrité |  |  | X | X |  |
| Confidentialité | X | X |  |  |  |
| Authentification |  | X |  | X |  |
| Non-répudiation |  |  |  |  | X |

## La sécurité sur Internet.

SSL // Secure Socket Layer.

TLS // Transport Layer Security.

Ensemble de protocoles permettant de sécuriser un paquet au niveau de la couche transport de TCP/IP.

Il y a comme une couche qui se rajoute entre la couche Application et Transport. Cette couche se sépare en 2 sous-couches. La couche « Handshake » qui regroupe le handshake (syn-ack) en tant que tel et qui établit la session. Le ChangeCipherSpec protocol qui s’occupe de la négociation des algorithmes en usage. Le Alert protocol pour la gestion des messages d’erreurs. Et la couche *Record* avec le *Record Protocol* qui s’occupe de la protection des données.

Présent en HTTPS, SSH. Pas un protocole de niveau application. Plutôt lent car une couche intermédiaire en plus.

IPsec // IPSecure

Ensemble de protocoles pour sécuriser un paquet au niveau de la couche réseau (OSI) / couche IP (TCP.IP). Remplacement de la couche internet de TCP/IP par la couche IPsec.

Mode transport // Les données transférées seulement qui sont chiffrées, authentifiées.

Mode tunnel // Totalité du paquet IP qui est chiffré, authentifié. Puis re-encapsulé dans un nouveau paquet.

# SSH

## Établissement d’une communication

1) Négociation Client-Serveur :

Les deux parties se mettent d’accord sur les différents algorithmes avec lesquels ils vont travailler. Notamment chiffrement symétrique, asymétrique, hachage. Une fois fait, le serveur propose sa KPUB si besoin est.

2) Authentification du serveur :

Le serveur doit répondre à un défi. Le client invente un défi, et l’envoie de manière chiffrée avec la KPUBs. Si le serveur est le bon, il possède la bonne KPRIVs pour déchiffrer le défi, et il le résout. Il envoie la réponse en retour au client. Le client vérifie et si c’est bon, alors OK. Grâce à cette étape, on peut confirmer l’objectif d’Authentification.

3) Établissement d’un canal sécurisé :

Le client et le serveur se mettent d’accord sur une clé de session de manière sécurisée (Diffie-Helmans). Le canal est créé.

4) Authentification du client :

2 options : Mot de passe ou Paire de clés.

- Par mdp, le client donne un login et un mdp et si le serveur reconnait cette combinaison et qu’elle existe, il peut rentrer. Chiffrement du login/mdp par du SHA et la clé de session.

- Par paire de clés, la connexion se déroule sans mdp ni login. Même principe que l’étape 2 mais dans l’autre sens. Le client a, au préalable, généré son trousseau de clé à lui et l’aura envoyé dans la home directory du serveur dans le fichier « authorized\_keys ». Le serveur créé un défi, le chiffre avec la KPUB et lui envoi. S’il est le bon client, il pourra déchiffrer avec sa KPRIV, répondre au défi et lui renvoie la réponse. Le serveur vérifie, et si tout OK, le user est connecté sans même rentrer un mot de passe.

Ces étapes permettent de cocher les objectifs d’Intégrité, de Confidentialité, et d’Authentification.

5) Authentification du serveur :

La communication peut démarrer sereinement. Elle se passera dans le canal ssh créé durant l’étape 3 et les données seront chiffrés grâce à la clé de session ainsi que du SHA.

## Fichier de conf

Configuration du SERVEUR : /etc/ssh/sshd\_config

Configuration du SERVICE : /etc/sysconfig/sshd

Démarrer, redémarrer, arrêt → systemctl stat/restart/stop sshd

Copier des fichiers → scp

Générer un trousseau de clés → ssh-keygen

Le service sshd possède son propre trousseau, avec 3 paires de clés par défaut.

## Implémentation par mot de passe.

Le serveur a 3 paires de clés par défaut dans le dossier → /etc/ssh. Du ecdsa, du ed25519 et du rsa. Si le client n’a pas encore la clé publique du serveur, il lui propose de la télécharger. Les clés publiques de serveurs connus → ~/.ssh/known\_hosts.

**ssh-keygen -t rsa -b 1024** // Génére des clés de type rsa et de longueur 1024 bits.

**ssh-keygen R 192.168.1.100** // Retire la clé correspondante a l’ip dans le fichier known\_host

Le fingerprint a pour but de confirmer l’authentification et non-répudiation. Lors de la première connexion, possible de voir ce fingerprint et de le comparer sur le serveur.

**ssh-keygen -lf « emplacement\_clé »** // Affiche l’empreinte d’une clé.

Le fichier known\_hosts peut apparaître dans plusieurs home directories et contenir des clés publiques de serveurs différents.

Historique de connexion dans /var/log.

## Implémentation par paire de clés.

Générer une clé côté client → **ssh-keygen -t rsa**

LA KPUB doit se trouver dans le dossier «home/.ssh/authorized\_keys » du serveur. Attention à l’endroit où sont placés les clés publiques des clients.

Envoyer sa clé sur l’utilisateur root du serveur → **ssh-copy-id -i «/root/.ssh/‘KPUB’ » « user@IPsrv »**

## Message d’erreur a la connexion sur un serveur quelconque : « REMOTE HOST IDENTIFICATION HAS CHANGED »

Signifie que la paire de clé du SERVEUR SSH a été regénéré et non celui de L’UTILISATEUR.

# Pratique

## Start / Stop / Restart

**- systemctl « x » sshd**

## Emplacement des trousseaux de clé SSH

**- /etc/ssh**

## Fichier de configuration du SERVICE SSHD

**- /etc/sysconfig/sshd**

## Fichier de configuration du SERVEUR SSHD

**- /etc/ssh/sshd\_config**

Possibilité de modifier le port d’écoute « **Port = xx** »

L’écoute de d’une famille d’ip via « **AddressFamily any / inet / inet6**»

La position des trouceaux de clé via le « **Hostkey …**»

Le temps maximum pour la connexion via ssh « **LoginGraceTime** »

Configurer la connexion via pair de clé ou password via « **Pubkey/PasswordAuthentication x**»

Permettre la connexion sans mot de passe via « **PermitEmptyPasswords x** »

Maintenir la connexion entre le client et le serveur via « **ClientAliveInterval/CountMax x**»

Empêcher quoiqu’onques de se connecter en tant qu’utilisateur « **DenyUsers**»

Autoriser des utilisateurs spécifiques « **AllowUsers x**»

Permettre à « root » de se connecter ou non « **PermitRootLogin x**»

Permettre à « root » de se connecter sans mdp et uniquement par pair de clé «**PermitRootLogin without-password** »

Permet l’utilisation du SFPT en enlevant le commentaire de **« Subsystem⬄sftp⬄/usr/libexec/openssh/sftp-server »**

## Autoriser la création d’un type de clé. SERVEUR SSHD

**- systemctl enable sshd-keygen@«type de clé».service**

## Stopper la création d’un type de clé. SERVEUR SSHD

**- systemctl mask sshd-keygen@«type de clé».service**

## Régénération des différents trouceaux SSHD :

- Méthode simple : suppression des trousseaux de clé dans le dossier /etc/ssh ensuite redémarrer le service ssh via systemctl restart sshd (Inconvénient : interruption du service ssh pendant quelque seconde)

-Méthode « moins simple » mais plus pro : utiliser la commande

**- ssh-keygen -t « type de clé » -N ’’ -f /etc/ssh/ssh\_host\_ « type de clé » \_key**

## Empreinte (Fingerprint) d’une clé.

- **ssh-keygen -lf /etc/ssh/ssh\_host\_«type de clé»\_key.pub**

## Copie de fichier

En local :

**- scp «fichier à copié» «userDest@ip:/dossierDest»**

En remote :

**- scp «userSource@ipsource:/dossierSource» «userDest@ipDest:/dossierDest»**

## Création trousseau de clé user

**- ssh-keygen -t «type de clé»**

## Copie clé publique user 🡪 serv.

**- ssh-copy-id -i «nom de la clée» «userServ@ipServ»**

## Emplacement des clés autorisé

**- ~/ssh/authorized\_keys**

## Fail Connexion après télécharger une clé

Vérifier via l’admin du serveur s’il a regénérer un nouveau trousseau de clé OU si elle a été détourner. S’il la regénérer il suffit d’aller supprimer la clé correspondante dans le dossier du point suivant.

## Fichier de stockage des clés connues

**- ~/ssh/known\_hosts**

# Espionnage de trame

## Utilisé Tshark pour espionner un échange ssh

**tshark -i « interface d’écoute » « protocole utilisé » port « n° de port »**

## Colorisation des flux de conversation sous SSH

Couleur Bleu = Serveur

Couleur Rouge = Client

# Tunneling SSH

## Différence Local / Remote

Toucher quelque chose grace au tunel : utiliser le LOCALHOST

|  |  |  |  |
| --- | --- | --- | --- |
|  | Service SSHD | Service SSH | Création du Tunnel |
| Local | Server | Client | Client |
| Remote | Client | Server | Admin Server |

## Local = CLIENT à 2 machines

**- ssh -N -f -L port\_client:ip\_serveur:port\_utiliser\_coté\_server utilisateur:ip\_serveur**

(-N = Aucune commande à faire // -f = mise en arrière-plan // -L = Local // port client = Port Client Libre // IP server // port utiliser : port utiliser pour la communication coté serveur // utilisateur = utilisateur présent et crée dans le serveur // IP server)

## Remote = SERVEUR à 2 machines

**- ssh -N -f -R port\_client:ip\_client:port\_utiliser\_coté\_server utilisateur:ip\_client**

(-N = Aucune commande à faire // -f = mise en arrière-plan // -R = Remote // port client = Port client Libre // IP client // port utiliser : port utiliser pour la communication coté server // utilisateur = utilisateur présent et crée dans le client // IP client)

## Local = CLIENT à 3 machines

**- ssh -N -f -L port\_client:ip\_serveur:port\_serv utilisateur\_intermédiaire:ip\_intermédiaire**

(-N = Aucune commande à faire // -f = mise en arrière-plan // -L = Local // port client = Port Client Libre // IP server // utilisateur intermédiaire : utilisateur crée sur l’intermédiaire // IP intermédiaire = Adresse IP de l’intermédiaire (Ou Nom de domaine))

## Remote = SERVEUR à 3 machines

**- ssh -N -f -R port\_client:ip\_serveur:port\_serv utilisateur\_client:ip\_client**

(-N = Aucune commande à faire // -f = mise en arrière-plan // -L = Remote // port client = Port Client Libre // IP server // utilisateur client : utilisateur crée sur le client // IP client = Adresse IP du client)

## Remarque

Le Natting / Forwarding DOIT être actif pour que la situation a 3 machines fonctionne correctement.

Port à utiliser toujours **Strictement Supérieur à 1023** car en-dessous ce sont des PORT dits connus et réserver au serveur.

# Le Natting / Forwarding

IP Forward → Sur un routeur, lorsqu’un paquet arrive sur une interface mais qu’il est destiné à une autre, le routeur « fait suivre » le paquet jusqu’au bonne endroit / transmission de paquets entre les différentes interfaces d’un routeur.

Port forwarding → Même principe mais pour les numéros de port / changement de point de vue sur la destination.

Le NATing doit également être activé car grâce au NAT, on effectue un changement d’adresse de source dans le paquet lui-même / lui permet de faire un changement d’adresse de source.

Table NAT = un port et une adresse d’une interface correspond à une adresse et un port d’une autre interface. L’IP et le numéro de port change dans le paquet lors du passage d’une interface à une autre.

Forwarding :

**- iptables -t nat -i PREROUTING -p qqc -i int\_wan --dport qqc -j DNAT --to ip\_serv :port\_serv**

(Début par défaut donc osef // -p qqc = QQC représente un protocole de communication utiliser // -i int\_wan = l’interface d’entrer sur le routeur coter WAN // --dport qqc = QQC représente le numéro du port utiliser en vis-à-vis de l’interface d’entrer coter WAN // -j DNAT --to ip\_serv:port\_serv = Permet le forward (-j DNAT) vers (--to) le serveur que l’on veut toucher grâce a son IP et a sont port)

Configurer l’ip forward

**- mcedit /etc/sysctl.conf** // Fichier a config

**- net.ipv4.ip\_forward = 1** // Ligne a rajouter

Ensuite terminer la config en relançant le noyaux linux.

**- sysctl -p**

Pour qu’au prochain redémarrage le service reste actif il suffit de taper la commande :

**- systemctl enable iptables.service**

Aperçu que l’ip\_forward est actif via la commande :

**- cat /proc/sys/net/ipv4/ip\_forward** // La réponses doit être 1

## Natting :

**- iptables -t nat -A postrouting - o interface\_nat -j MASQUERADE**

Message entre client et intermédiaire sécuriser mais pas entre l’intermédiaire et le serveur a toucher.

Une configuration a 3 machines permet de contourner des règles de Firewalling.

## Commande supplémentaire

Visualiser la table Nat via la commande :

**- iptables -L -t nat**

On se connectera via la commande SSH :

**- ssh -p qqc ip\_WAN //** (-p qqc = Port coter WAN tels que l’on a spécifier dans la commande au-dessus // ip\_wan = adresse IP de l’interface coté Wan spécifier au-dessus

**- ss -tnl** // Permet de voir les ports en cour d’utilisation.

**- ss -tn** // Affiche les connections active/établie

**- mcedit /etc/ssh/ssh\_config //** Configuration client SSH et « AddressFamilly » permet d’envoier des demande ssh en ipv4/6, pour seulement mettre ipv4 il faut mettre le « any » en « inet »

**- mcedit /usr/lib/systemd/system/telnet.socket** // Modifie le service telnet, pour uniquement ipv4 mettre : « ListenStream=0.0.0.0 :23 »

**- ps ax + kill l’id du process correspondant à la commande de création du canal.** // Permet de stopper le tunneling.

**-** **scp « fichier a envoyé » « user@IPSERV »:« dossier de réception »** // copier de ma machine sur un serveur.

**- scp « user@IPSERV »:« fichier a envoyé » « user’@‘IPSERV »:« dossier de réception  »** // copier de n’importe quelle machine sur n’importe quel serveur.

**- sha256sum « fichier a haché »** // Hachage d’un fichier pour avoir son empreinte.

**- crontab -e** // pour lancer des évènements à horaire fixe.

**- chmod +x « nom\_script »** // rendre un fichier exécutable.

**- ./« nom\_script »** // exécuter un script.

**- echo $1 | passwd --stdin $1** // donner le mdp à l’utilisateur créer avec le paramètre.

**- echo $1 | passwd --stdin root** // changer le mdp de root avec le paramètre.