Dossier

Avantages et faiblesses du protocole SSH

Andrew Faulkner, Florian Blanc

Degré de difficulté

Cet article s'adresse aux utilisateurs souhaitant tout d'abord comprendre le fonctionnement du protocole SSH et l'implémenter sur leur réseau afin d'en tirer profit. Nous aborderons ici les différents aspects du protocole SSH, ses fonctionnalités, ses avantages et ses faiblesses sans oublier les erreurs à éviter pour ne pas compromettre son serveur.

Le protocole *SSH* pour *Secure Shell* est largement utilisé de nos jours visant à garantir l'authenticité, la confidentialité et l'intégrité des données. En utilisant *SSH*, la transmission des informations au sein d'un réseau est chiffrée et ne peut donc, en théorie, être que difficilement compromise. SSH se décline en deux versions. La version 1 a été développée en 1995 par le finlandais *Tatu Ylönen* dans le but de sécuriser les connexions distantes. Aujourd'hui, la version la plus récente du protocole est la version 2, normalisée par *l'Internet Engineering Task Force* (*IETF*) en janvier 2006, apportant quelques corrections au niveau de la sécurité par rapport à *SSH-1* (notamment au niveau des algorithmes utilisés).

SSH est à la fois un protocole et un ensemble de programmes. Il possède plusieurs fonctionnalités. Il permet par exemple d'établir des sessions sécurisées sur un ordinateur distant, de transférer des fichiers sécurisés ou d'établir des tunnels sécurisés. Pour ce qui concerne l'établissement de communications sécurisées, *SSH* se base sur le protocole *SOCKS v5* intégré dans la plupart des équipements réseau. *SSH* apporte donc une amélioration indéniable aux protocoles non sécurisés tels que Telnet, Rlogin, FTP...

Rappel sur les fonctionnalités des différents algorithmes employés par SSH

Tout d'abord, il convient de rappeler les différents algorithmes utilisés par *SSH*.

En effet, *SSH* utilise deux algorithmes pour assurer l'authentification, *RSA* et *DSA*, ce dernier ne fonctionnant qu'avec la version 2 du protocole. Pour chiffrer les données, *SSH-1* utilise les algorithmes *DES*, *3DES*, *IDEA* ou

Cet article explique...

- Comment fonctionne le protocole SSH, son implémentation dans une architecture sécurisée et enfin comment sécuriser son serveur afin d'éviter tous types de compromissions.

Ce qu'il faut savoir...

- Vous devez avoir des notions de réseaux.
- Connaître le modèle OSI et le fonctionnement de TCP/IP.

Blowfish tandis que *SSH-2* emploie *AES*, *3DES*, *Blowfish*, *Twofish*, *Arcfour* ou *Cast128*.

Enfin, les fonctions de hachage sont assurées par les algorithmes *MD5* ou *SHA-1*.

Pour comprendre les différents algorithmes employés par SSH, il peut être utile de rappeler quelques principes de cryptographie.

À la base, la cryptographie se définit comme *l'étude des méthodes visant à transmettre des données de manière confidentielle*.

Aujourd'hui, cette définition a évolué et la cryptographie moderne cherche globalement à résoudre les problèmes liés à la sécurité des communications.

Elle vise ainsi à garantir des services de sécurité (comme la confidentialité des données, l'authenticité du document, l'authentification mutuelle et l'échange des clés de session) qu'elle met en œuvre au moyen de mécanismes de sécurité (chiffrement, signature, protocoles d'authentification avec échange de clés) basés sur des algorithmes cryptographiques.

Nous allons décomposer ce chapitre en 4 parties correspondant aux différents services et mécanismes de sécurité ainsi que la notion de certificats.

La confidentialité

Vous avez vu que la cryptographie cherche à transmettre des données de manière confidentielle. Pour cela, elle fait appel à la notion de chiffrement. Il existe deux différents types d'algorithmes cryptographiques : les algorithmes symétriques (appelés également à clé secrète) et les algorithmes asymétriques (appelés également à clé publique).

Les algorithmes symétriques utilisent une clé (dite de clé de session) afin de chiffrer le message. Cette même clé sera ensuite utilisée pour déchiffrer ledit message.

On distingue deux différents types de chiffrement symétriques : les algorithmes de chiffrement symétrique en continu (*stream cipher*) dont les plus utilisés sont *RC4*, *RC5* ; et les algorithmes de chiffrement symétrique par blocs (*block cipher*).

Les algorithmes de chiffrement symétrique par blocs les plus utilisés sont : *DES*, *3DES*, *IDEA*, *Blowfish*, *Cast128*, *AES*.

Ce type de cryptographie (symétrique) a l'avantage d'être rapide. Cependant, si la clé secrète est récupérée par une tierce personne, cette dernière pourra déchiffrer les messages (Figure 1).

Dans le cadre de la cryptographie asymétrique (dite à clé publique), les clés de chiffrement et de déchiffrement sont distinctes et non déductibles. Ainsi, connaître la clé publique ne permet pas de retrouver la clé privée correspondante.

La cryptographie asymétrique assure la confidentialité des données, mais peut également permettre (selon l'algorithme employé) de signer des messages.

Voici comment cela fonctionne : la clé publique sert au chiffrement et peut être utilisée par tout le monde pour chiffrer, mais seule la clé privée correspondante sera en mesure de déchiffrer le message, la confidentialité est donc assurée. Pour la signature, la clé privée est utilisée pour chiffrer et la clé publique, mise à disposition de tous, pour déchiffrer. La signature d'un document par un correspondant garantit mathématiquement que ce dernier en est l'émetteur.

Ainsi, la clé privée signe et la clé publique vérifie, garantissant l'identité de l'émetteur. Les algorithmes utilisés pour la cryptographie à clé publique sont basés sur des problèmes mathématiques difficiles à résoudre comme RSA (basé sur la factorisation des grands entiers) ou Elgamal (problème du logarithme discret). L'inconvénient des algorithmes actuels réside dans le fait qu'ils

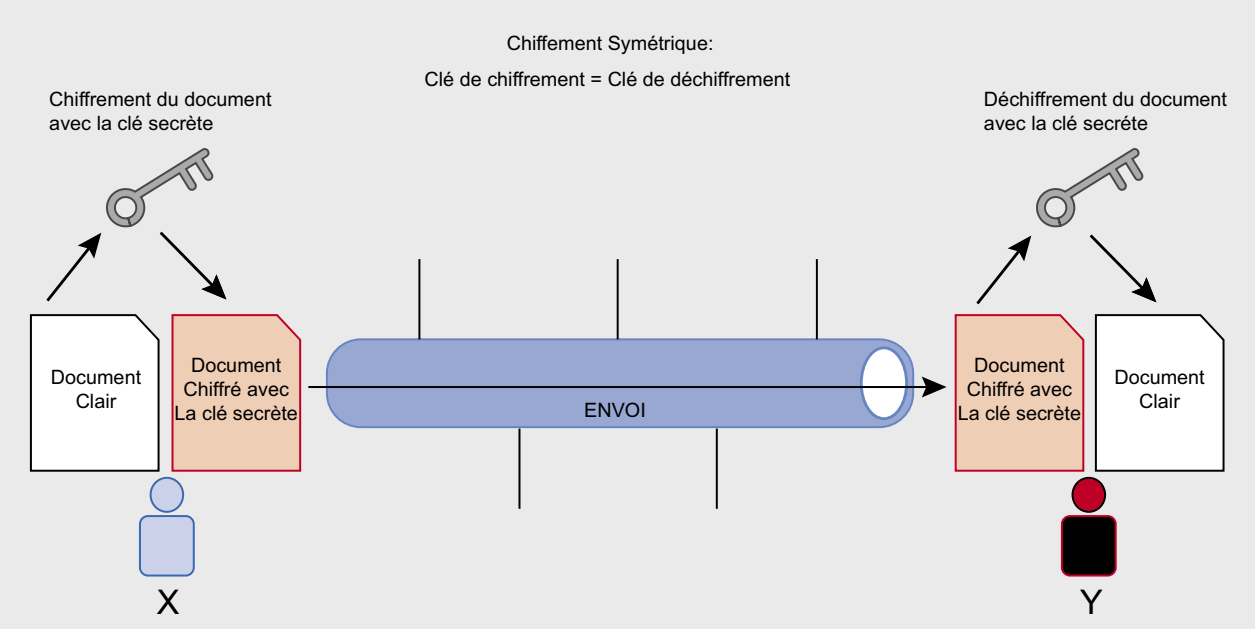

Figure 1. *Chiffrement symétrique*

sont plus lents que les algorithmes symétriques à clé secrète.

Ainsi, ils sont souvent utilisés pour chiffrer une clé de session secrète. Selon l'algorithme utilisé, ce dernier ne sera utilisé que pour chiffrer ou que pour signer un message. Malgré tout, certains algorithmes comme RSA ou ElGamal, permettent à la fois de chiffrer et signer.

Les inventeurs du concept de cryptographie à clé publique (inventé en 1976) sont Diffie & Hellman et les algorithmes les plus connus sont Diffie-Hellman et RSA (1978) (Figures 2 et 3).

Comme nous l'avons vu, la cryptographie asymétrique se révèle plus sûre que la cryptographie symétrique. Cependant, elle reste plus lente, cela est dû à la taille des clés utilisées afin de garantir la sécurité des données (par exemple RSA 2048 bits je vous laisse imaginer...). Afin de résoudre ce problème, il est possible de combiner les deux systèmes. Admettons que l'on veuille envoyer un document à un ami. Une fois le document rédigé, on le chiffre avec une clé secrète (algorithme symétrique, par exemple AES, pour assurer la confidentialité).

Puis, on chiffre la clé secrète avec la clé publique du destinataire (on est alors sûr que ce sera notre ami détenteur de la clé privée associée qui lira le document). Enfin, on envoie notre message. Notre ami

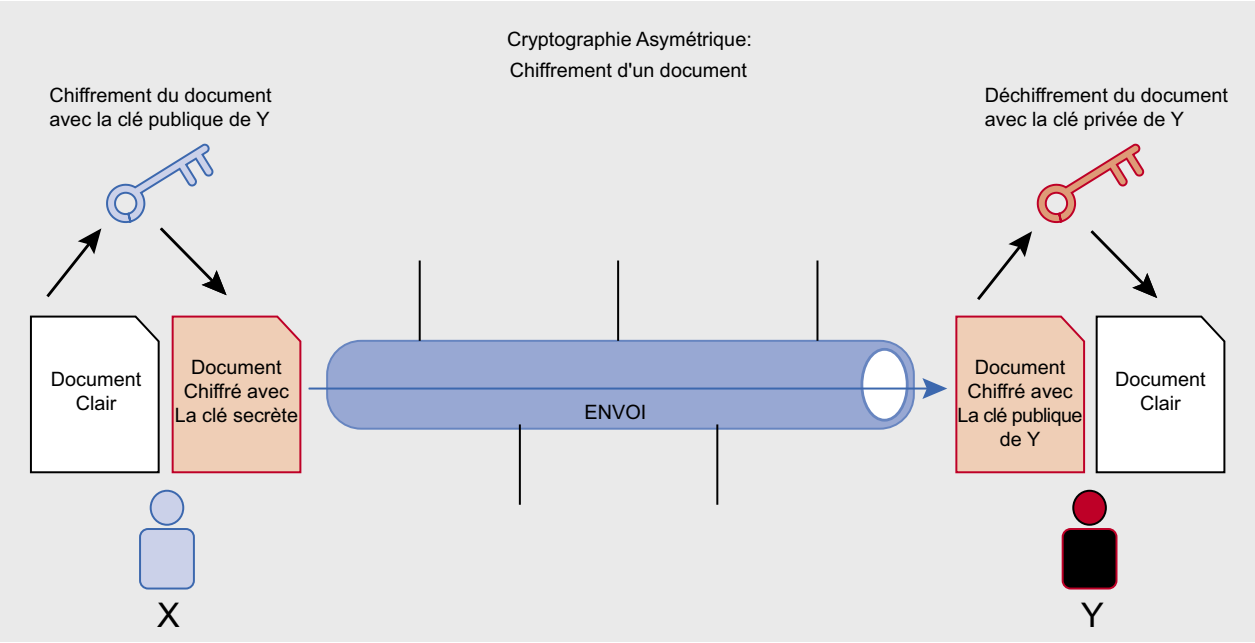

Figure 2. *Cryptographie asymétrique : chiffrement d'un document avec la clé publique Y*

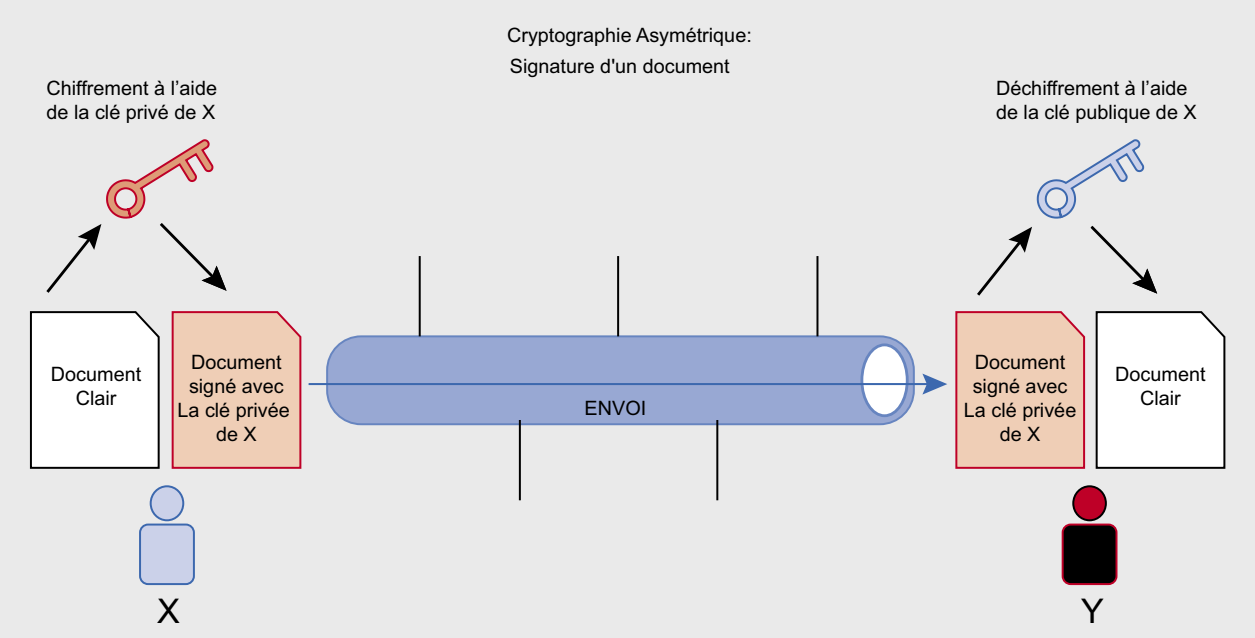

Figure 3. *Cryptographie asymétrique : chiffrement d'un document avec la clé privé de X*

reçoit le message, déchiffre la clé secrète à l'aide de sa clé privée. Puis avec la clé secrète, il déchiffre le document (Figure 4).

La signature numérique (norme ISO 7498-2)

Le but ici est de s'assurer de l'authenticité du document. L'authenticité est définie par l'authentification de l'émetteur du document (être certain de l'identité de l'auteur) et par l'intégrité du document (s'assurer que le document envoyé n'a pas été modifié durant l'envoi). La méthode employée afin d'assurer l'authenticité du document est la fonction de hachage à sens unique. Ce mécanisme garantit à la fois l'authentification de l'origine des données et l'intégrité.

Une fonction de hachage permet de hacher des données et donc de produire ce que l'on appelle une empreinte des données hachées (on parle également de résume ou de digest). L'empreinte obtenue est représentative du document haché. Ainsi, imaginons que nous souhaitions envoyer un document à un ami. Nous allons dans un premier temps hacher le message ensuite nous enverrons le document accompagné de l'empreinte obtenue (issue de la fonction de hachage). Notre ami, dès réception du document et de l'empreinte, va à son tour hacher le document.

Enfin, il va comparer l'empreinte obtenue avec celle reçue. Si les empreintes sont identiques alors l'intégrité du document est assurée, dans le cas contraire cela voudrait dire que le document a été modifié pendant sa transmission. Cependant, imaginons qu'une personne malveillante intercepte l'empreinte. Cette personne pourrait modifier l'empreinte sans modifier le document.

Le document serait alors systématiquement considéré comme altéré lors de sa réception et donc détruit. L'émetteur ne pourrait alors plus envoyer de document. D'autre part, seule l'intégrité du document est vérifiée. C'est pourquoi l'émetteur va chiffrer l'empreinte à l'aide de sa clé privée (le destinataire pourra la déchiffrer grâce à la clé publique associée). Cela va alors à la fois garantir l'intégrité du message, mais

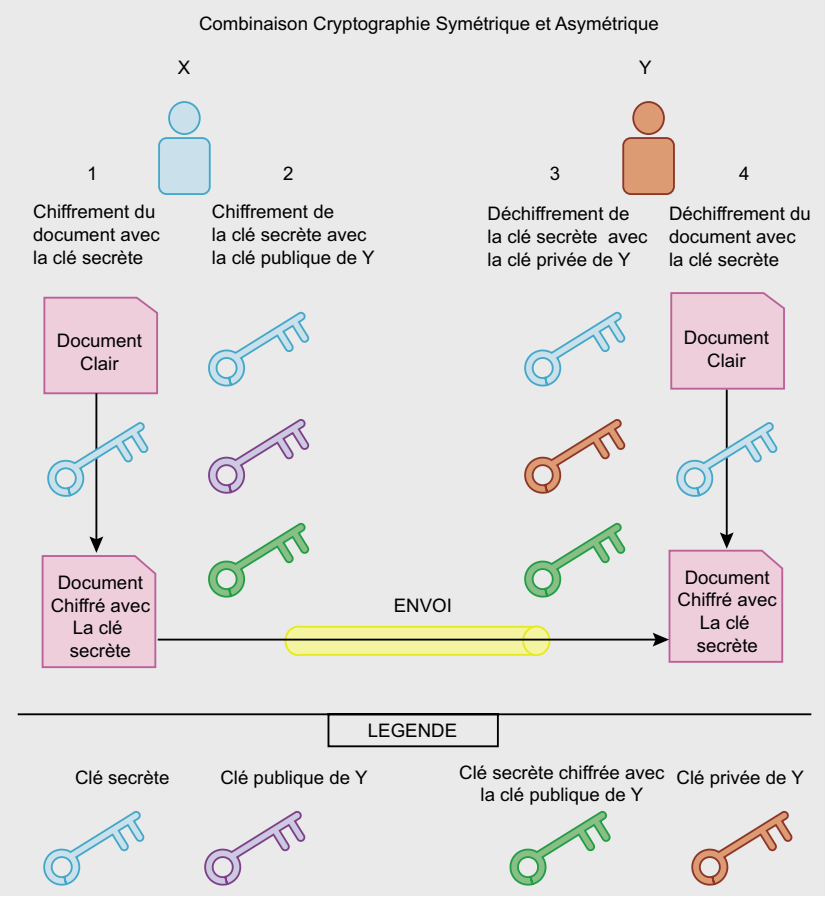

Figure 4. *Combinaison Cryptographie symétrique asymétrique*

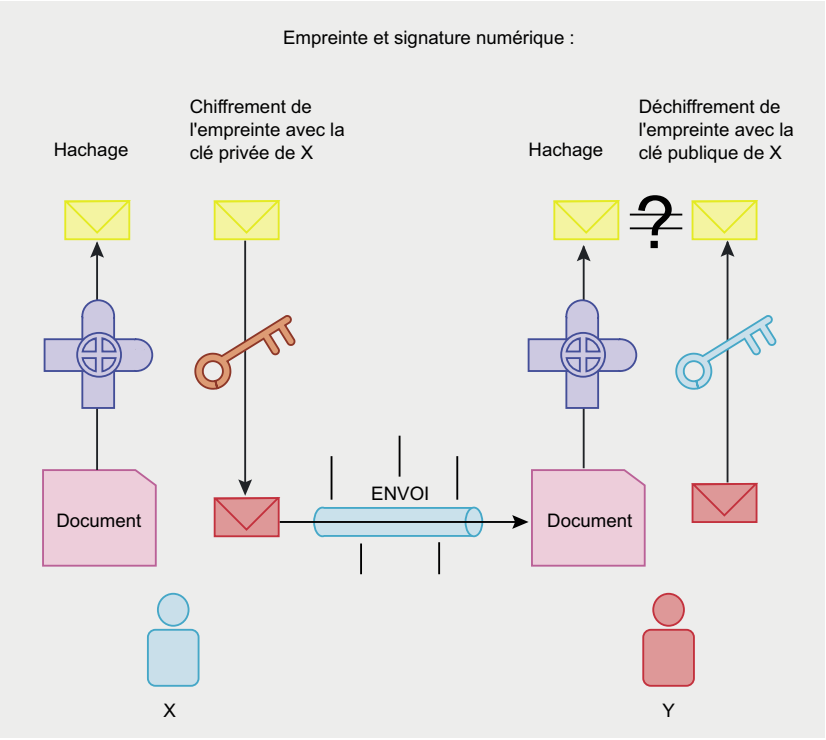

Figure 5. *Empreinte et signature numérique*

aussi l'authentification de l'émetteur. Il existe différents algorithmes de hacking. Les plus utilisés sont les suivants *MD5*, *SHA-1*, *SHA-2* et *HMAC*. La signature numérique assure donc l'authentification de l'origine des données, l'intégrité et la non-répudiation. Son fonctionnement se résume comme suit : on calcule l'empreinte du message à signer, puis on chiffre cette empreinte. Voici les algorithmes utilisés : *DSS* (Combinaison de *SHA* avec *El-Gamal*) et *RSA* (combinaison *MD5* ou *SHA* avec *RSA*) (Figure 5).

Échange de clés et authentification

L'échange de clés et l'authentification font parties de ce que l'on appelle les protocoles cryptographiques. Le but est d'établir une relation de confiance entre les participants.

Il existe deux méthodes pour établir les clés. La première est le transport. Dans ce cas de figure, une clé de session est générée aléatoirement et on la chiffre à l'aide d'un algorithme à clé publique. Une fois la clé de session chiffrée, on l'envoie au destinataire. Un exemple de transport est RSA (voir illustration des combinaisons symétriques et asymétriques pour comprendre le principe de chiffrement d'une clé de session).

La seconde méthode est la génération de clé (par exemple le protocole *Diffie-Hellman*) qui consiste à établir un secret partagé à partir d'informations publiques sans pour autant connaître des informations préalables entre les deux participants. Voici la base du fonctionnement du protocole *D&H* en prenant deux participants X et Y :

- X et Y choisissent un entier *n* premier (public) et un entier *g* (public aussi) primitif par rapport à *n*.
- X choisit un nombre entier a (gardé secret) à partir duquel il calcule la valeur publique `A = ga mod n`. Il envoie ensuite à Y : g, n, A.
- Y calcule alors `B = gb mod n` puis envoie la valeur publique *B* à X.
- X est en mesure de calculer `KAB = Ba mod n` - Y est en mesure de calculer `KBA = Ab mod n`.

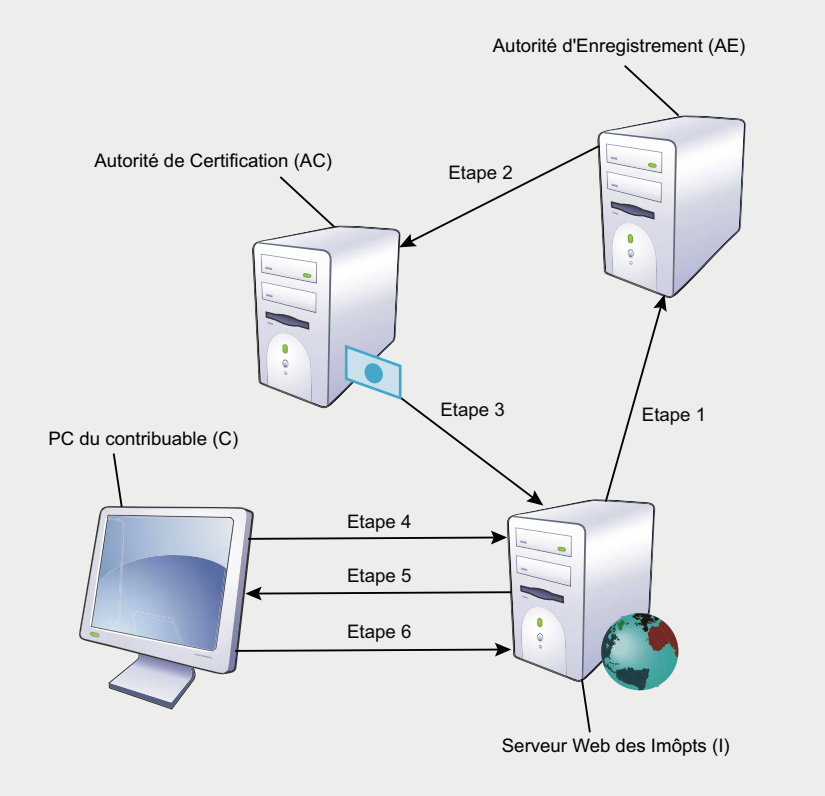

Figure 6. *Déclaration des impôts*

Figure 7. *Demande de certification*

- `KAB = KBA = gab mod n` constitue le secret partagé entre X et Y.
- *a* et *b* étants secrets, même si une personne écoute la communication, elle ne pourrait pas calculer `gab mod n`.

Exemple en chiffres Soient *n* = 23, *g* = 3, *a* = 6, *b* = 15 :

```
A = 36 mod 23 = 16
B = 315 mod 23 = 12
```

Clé secrète :

```
126 mod 23 = 1615 mod 23 = 9
```

Limite du protocole D&H

Si une personne écoutait la communication, elle connaîtrait *g*, *n*, *A*, *B* mais à aucun moment elle ne connaîtrait `gab mod n` (à moins de résoudre les logarithmes de *A* et *B* pour retrouver *a* et *b*). Par contre, il est possible de réaliser une attaque de l'homme du milieu. En effet, en se plaçant au milieu de l'attaque entre X et Y, et en interceptant *ga* et *gb* il nous sera possible de réaliser une attaque de type *Man In The Middle*. Autrement dit, on intercepte la clé *ga* envoyée par X et on envoie à Y non pas *ga* mais *ga'*. On fait de même pour *gb* que l'on modifie pour *gb'* et que l'on envoie à X. Nous sommes alors en mesure de communiquer avec X et Y en utilisant respectivement les clés partagées *gab'* et *ga'b* et nous avons leurrés X et Y qui ont échangé une clé secrète non pas entre eux mais avec nous.

Une solution est d'authentifier les interlocuteurs, ce que l'on appelle

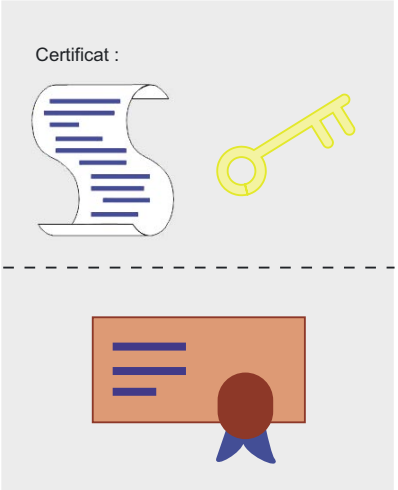

Figure 8. *Création de la certification*

Diffie-Hellman authentifié, en utilisant les certificats (cela nécessite alors l'implication d'une tierce personne fiable) ou en utilisant le mécanisme de signature. Une relation de confiance est alors établie avant que les participants ne se connaissent.

Les certificats (norme X.509)

Les différents types de cryptographie ne sont que des outils et une bonne gestion des clés publiques dans le cadre de la cryptographie asymétrique (pour d'importantes infrastructures par exemple) est impérative. Il existe donc des infrastructures à clés publiques (PKI) chargées de mieux gérer les clés publiques. Ces systèmes se basent dans la plupart des cas sur ce que l'on appelle les certificats numériques. Les certificats numériques peuvent être comparés à des cartes d'identité. Ils permettent d'authentifier les personnes qui vont correspondre. Un certificat contient différentes informations relatives à l'identité d'une personne, sa clé publique ... Celui qui détient le certificat est en fait le possesseur de la clé privée associée au certificat. Un certificat est délivré par une autorité de certification (CA) et est normalisé. Cependant, des certificats falsifiés peuvent être délivrés par des personnes malveillantes il faut donc rester vigilant. Les certificats permettent de distribuer les clés de manière sécurisée (authentification et intégrité assurées) et de les stocker de façon sûre. Un certificat est une structure de données qui lie la clé publique à différents éléments par l'intermédiaire de la signature d'une autorité de confiance. Il contient comme informations le nom du propriétaire de la clé, la clé publique, les dates de validité, les droits, un numéro de version (exemple X, 509 Version 3) un numéro de série... Le certificat est émis par une *Autorité de Certification* (CA) garantissant l'exactitude des données. Il existe des listes de révocation (CRL) permettant de révoquer des certificats avant expiration.

Exemple d'utilisation de certificats numériques

La déclaration des impôts (aperçu simplifié). Il est désormais possible en France de déclarer ses impôts sur Internet. Cela pose donc un enjeu de sécurité extrêmement important. Pour assurer la sécurité, la déclaration est effectuée par l'utilisation des certificats numériques. Pour cela, le contribuable doit installer auparavant le certificat proposé sur le site de l'administration des impôts. *But* : Le contribuable C veut déclarer ses impôts en ligne. Il faut donc que ce dernier soit sûr que le destinataire est bien le site des impôts et non un site pirate. D'autre part, l'envoi de la déclaration doit être sécurisé (chiffrement des données) voir Figure 6.

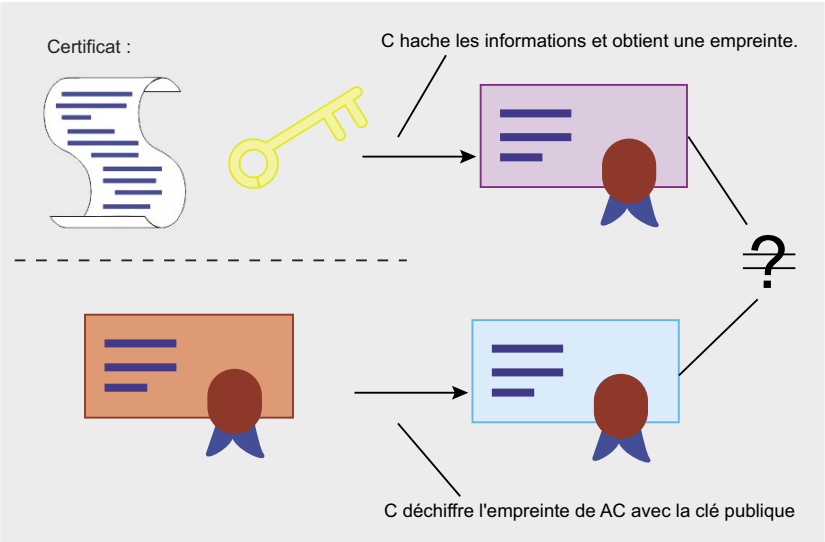

Figure 9. *Vérification du certificat*

Décomposition en 6 étapes

Voici en en détail les étapes.

- *Étape 1* : Demande de certification. Le site des impôts (*I*) fait une demande de certification à *l'Autorité d'Enregistrement* (*AE*). I fournit les informations nécessaires à son identification (exemple : nom de l'entreprise, adresse du site Internet...) (Figure 7),
- *Étape 2* : Vérification des informations et envoi à *l'Autorité de Certification* (*AC*). *AE* vérifie l'exactitude des informations fournies. Si ces dernières sont correctes, *AE* valide puis envoie à *AC*,
- *Étape 3* : Vérification et ajouts d'informations par *AC*. *AC* vérifie que *AE* a bien validé les informations, puis il rajoute ses propres informations au document et enfin hacke le tout. Il chiffre ensuite l'empreinte obtenue avec sa clé privée, ce qui constitue la signature numérique. Le certificat est alors formé. *AC* envoie le tout au demandeur de certificat (*I*). *Bilan* : Le certificat est créé, il contient : Les informations concernant l'identification du propriétaire du certificat (*I*) ainsi que sa clé publique. Et contient aussi la durée de validité du certificat et l'empreinte chiffré des informations signée par *AC* (Figure 8),
- *Étape 4* : Téléchargement du certificat. Le *contribuable* (*C*) se rend sur le site des *impôts* (*I*), s'identifie en renseignant son numéro fiscal, puis télécharge le certificat,

- *Étape 5* : Vérification du certificat. Le *Contribuable* (*C*) vérifie les informations contenues dans le certificat. Il hack ensuite ces informations puis déchiffre l'empreinte (réalisée par *AC*) avec la clé publique de *AC*. Enfin, il compare le résultat des deux empreintes obtenues (Figure 9),
- *Étape 6* : Envoi de la déclaration d'impôts. Une fois que la relation de confiance est établie entre le contribuable et le site des impôts, ce dernier peut envoyer en toute sécurité sa déclaration (qu'il aura auparavant chiffrée à l'aide de la clé publique de *I*). Le site des impôts (*I*) déchiffre ensuite la déclaration avec sa clé privée.

Approfondissement du protocole SSH Version 2

En premier lieu, il est impératif que SSH soit implémenté dans une architecture sécurisée. Par exemple, si le serveur SSH est configuré pour une authentification de type *login*/*password*, il convient de différencier le mot de passe du serveur SSH des autres mots de passe habituellement utilisés. En effet, si une personne vient à trouver un couple *login*/*password* d'un serveur Y et que ce même couple est identique à celui utilisé pour le serveur SSH elle pourrait alors se connecter au serveur SSH... De plus, si une authentification par clé publique est instaurée et que le client dont la clé publique est autorisée est compromis, alors la personne pourrait également se connecter (d'où l'importance de la *passphrase* que l'on verra plus loin dans cet article).

SSH s'appuie sur une architecture de type *client-serveur* pour assurer l'authentification des entités communicantes, la confidentialité et l'intégrité de leurs données transmises sur un réseau.

Pour rappel

L'authentification de la source prouve que les données proviennent bien de l'émetteur déclaré. Généralement le couple *Utilisateur*/*Mot de passe* authentifie l'émetteur, cependant, contrairement à telnet où les informations sont transmises en clair sur le réseau, SSH chiffre le trafic durant l'authentification grâce à *RSA* ou *DSA*.

La confidentialité des données certifie que seules les personnes autorisées peuvent accéder aux données en clair, pour cela *SSH* a recours aux algorithmes de chiffrements symétriques *3DES*, *Blowfish*, *Twofish* ou encore *Arcfour* et *Cast128*.

Enfin, l'intégrité des données est assurée par les fonctions de hachage *MD5*, *SHA-1*.

Le protocole *SSH* se décline, aujourd'hui, en 2 principales versions, la première, *SSHv1* étant aujourd'hui délaissée au profit de *SSHv2*.

Cela tient, d'une part, à ce que *SSHv1* est monolithique, ce qui signifie qu'il est constitué d'un bloc et que toutes améliorations ou modifications pouvant être apportées à *SSHv1* se heurtent à des problèmes d'incompatibilité et rendes les changements complexes. *SSHv2* quant à lui, est modulable : étant divisé en trois couches, il est plus facile de modifier l'une des couches sans affecter les deux autres ce qui offre une meilleure interopérabilité.

Cela tient, d'autre part, à ce que *SSHv1* ne permet d'établir qu'un seul et unique canal par session contrairement à la version 2 où il est possible de crée une multitude de canaux par session.

Les autres différences sont au niveau sécurité. Par exemple, pour contrôler l'intégrité des données, *SSHv1* utilise *CRC32* qui est cryptographiquement peu fiable. Dans la version 2, *HMAC* est employé. De plus, dans *SSHv1*, le tunnel est chiffré via une clé de session (chiffrement symétrique donc). Cette clé de session est transmise par le client, elle est valable pour la durée de la session et est unique pour les deux sens. *SSHv2* apporte ici une plus grande sécurité : les clés de session sont négociées (protocole *Diffie-Hellman*), les négociations au niveau de l'établissement du tunnel sont également plus pointues (entre autres). Enfin, une autre différence entre les deux versions se situe au niveau des algorithmes utilisés. *SSHv2* propose d'autres algorithmes en plus de ceux supportés par *SSHv1* (Figure 10).

- `PACKET LENGTH` : taille du paquet ssh (sans tenir compte des champs `PACKET LENGTH` ni `MAC`).
- `PADDING LENGTH` : correspond à la taille en octets du champ `RANDOM PADDING`.
- `PAYLOAD` : contient les données importantes du paquet (peut éventuellement être compressé par *zlib*).
- `RANDOM PADDING` : bourrage aléatoire de manière à ce que les 4 premiers champs du paquet (à savoir tous sauf `MAC`) soient un multiple de 8 ou un multiple du bloc de chiffrement (selon les résultats du multiple de 8 et du multiple du bloc

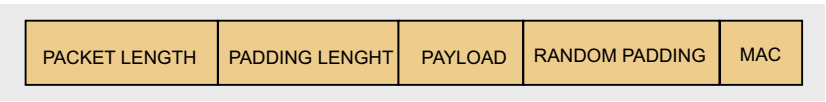

Figure 10. *Structure d'un paquet*

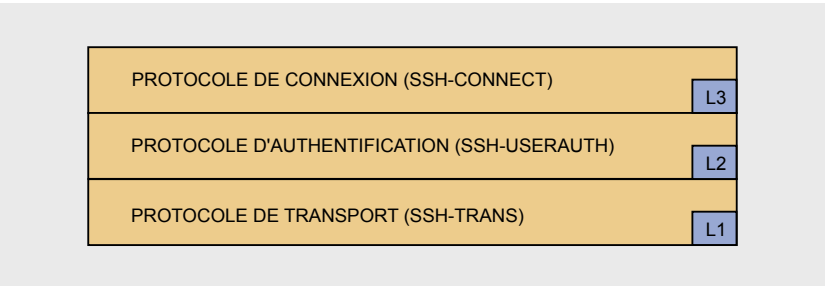

Figure 11. *Les trois couches SSHv2*

de chiffrement, le plus grand sera gardé).

- MAC : *Message Authentication Code*. Contient les octets du MAC si jamais il a été négocié (Figure 11).

Tout d'abord, commençons par la couche transport sur laquelle les deux autres s'appuient. La couche transport repose sur *TCP/IP*. SSH-TRANS est définie par la *RFC 4253*.

Cette couche fournit la confidentialité des données (chiffrement), elle permet l'authentification du serveur, elle assure l'intégrité des données et peut permettre de compresser les données. C'est à cette couche que sont négociés la méthode d'échange des clés utilisés, les algorithmes asymétriques et symétriques ainsi que les algorithmes d'authentification et de hacking.

Ensuite vient le protocole d'authentification : SSH-USERAUTH. Ce dernier fonctionne, comme vous l'avez compris, au-dessus de SSH-TRANS (ce qui suppose que le protocole sous-jacent assure l'intégrité et la confidentialité des données. SSH-USERAUTH fournit un tunnel unique authentifié pour la connexion *SSH* (SSH-CONNECT). Il a donc pour but de garantir l'identité du client auprès du serveur.

Enfin, il reste la couche connexion (SSH-CONNECT) qui fournit aux clients un certain nombre de services tout en se servant de l'unique tunnel créé. Le protocole de connexion *SSH* permet par exemple d'ouvrir des sessions shells interactives ou de transférer les applications *X11*. Tous ces canaux sont donc multiplexés en un seul tunnel chiffré.

Nous pouvons à partir de ces informations comprendre un des intérêts de *SSH*. En effet, il est possible d'encapsuler des protocoles applicatifs non sécurisés dans *SSH* afin d'apporter aux applications des services de chiffrement et d'intégrité de manière transparente.

Ainsi, SSH permet de sécuriser les données transitant sur le réseau grâce

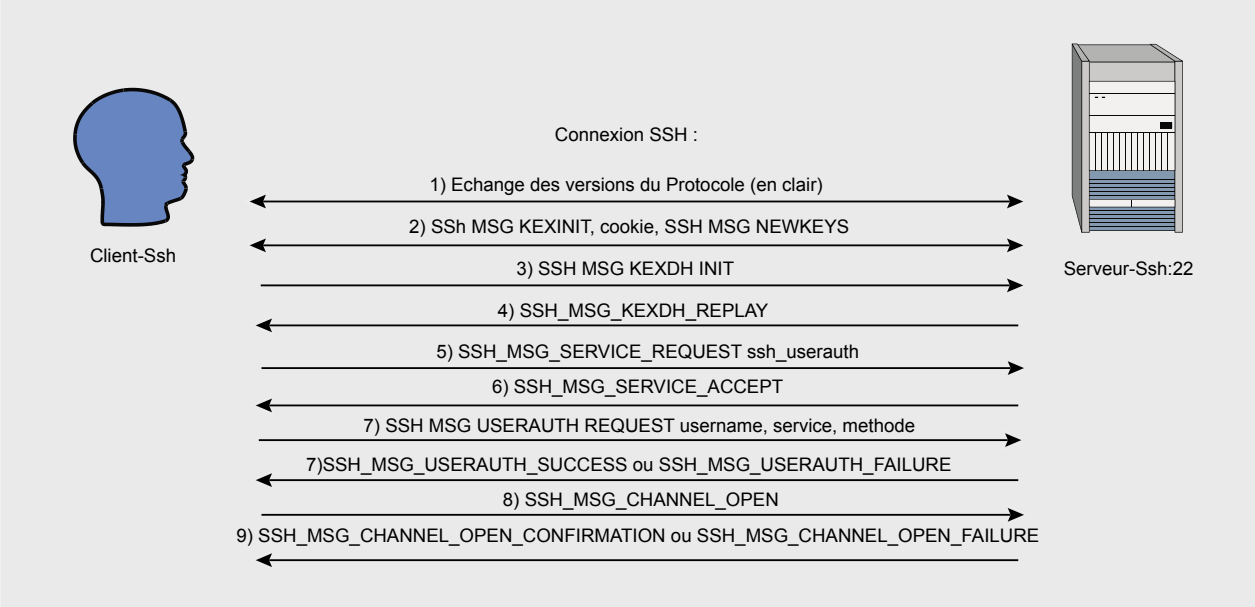

Figure 12. *Connexion SSH*

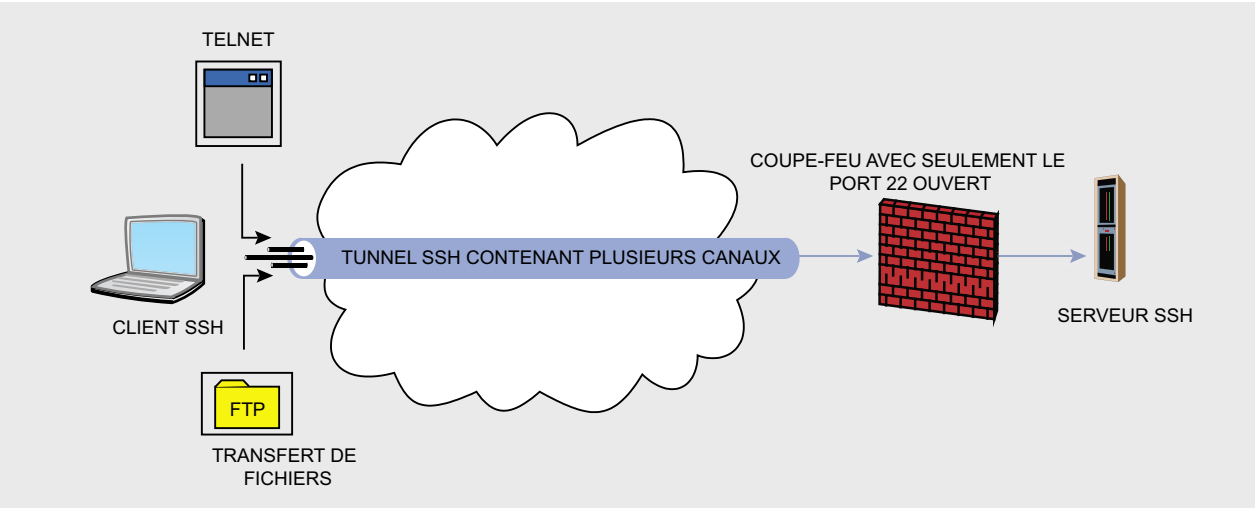

Figure 13. *Tunnel SSH*

à ses mécanismes de chiffrements, et est capable à la fois d'assurer l'intégrité des données et de garantir leur non-répudiation.

Maintenant que nous avons vu à quoi correspondent les différentes couches implémentées dans SSHv2, nous allons décomposer une connexion SSH afin de mieux comprendre les principales étapes nécessaires à son initialisation.

Par défaut, le serveur écoute sur le port *TCP* numéro *22* (attribué par *l'IANA*). Dès que la connexion est établie, le client et le serveur échangent les versions du protocole *SSH* utilisées ainsi que la version logicielle. Cet échange a la particularité de se faire en clair vu qu'aucun échange de clés n'a encore eu lieu.

Le serveur envoie ensuite le message `SSH_MSG_KEXINIT`. Ce message liste les algorithmes supportés pour le chiffrement, le hacking... et fournit également un cookie sur 64 bits à retourner. Le client répond en indiquant les méthodes supportées et renvoie une copie du cookie. Pour clore l'échange, le message `SSH_MSG_NEWKEYS` est envoyé.

La prochaine étape consiste à échanger les clés publiques *D&H* et à authentifier le serveur. Pour cela, le client envoie le message `SSH_MSG_KEXDH_INIT` et le serveur répond avec le message `SSH_MSG_KEXDH_REPLY`.

Le client demande ensuite un service par le message `SSH_MSG_SERVICE_REQUEST` qui correspond à l'activation de la couche suivante. Par exemple il envoie `SSH_MSG_SERVICE_REQUEST` ssh_userauth pour initialiser la couche `SSH-USERAUTH`. Le serveur accepte en renvoyant le message : `SSH_MSG_SERVICE_ACCEPT` (dans ce message les méthodes d'authentification supportées par le serveur sont exposées).

Maintenant que la couche d'authentification est activée, le client fait une demande d'authentification proposant une des méthodes d'authentification : `SSH_MSG_USERAUTH_REQUEST` username, service, methode... Le serveur y répond, acceptant ou non la méthode demandée : `SSH_MSG_USERAUTH_SUCCESS` ou `SSH_MSG_USERAUTH_FAILURE` selon les cas.

Si le client souhaite ensuite, par exemple, faire de la redirection de flux X11, il commence par envoyer un message de type `SSH_MSG_CHANNEL_REQUEST`. Ensuite, il envoie un message `SSH_MSG_CHANNEL_OPEN` pour ouvrir le canal X11. Le serveur peut alors répondre en confirmant l'ouverture du canal (`SSH_MSG_CHANNEL_OPEN_CONFIRMATION`) ou en retournant une erreur comme suit : `SSH_MSG_CHANNEL_OPEN_FAILURE` (Figure 12).

SSH, comme nous l'avons vu, permet de sécuriser n'importe quel protocole applicatif grâce à ses mécanismes de *port forwarding* et de *tunneling*. Cela fonctionne comme suit (pour l'implémentation voir plus bas) (Figure 13). Pour une redirection de port (port forwarding) voir Figure 14.

Les illustrations sont assez explicites. Dans le premier cas, le serveur *SSH* fait également office de serveur telnet (seulement pour notre exemple, car dans la réalité ce ne serait vraiment pas ingénieux...) et de serveur *ftp*.

Dans le second cas, les serveurs *ftp*, *pop* et *SSH* sont distincts et donc les flux sont redirigés vers le serveur concerné.

Dans tous les cas, ce genre d'implémentation a deux avantages importants. Le premier avantage est de limiter l'ouverture des ports sur le garde-barrière. En effet, la probabilité de compromettre un poste est d'autant plus importante que le nombre de ports ouverts sur ce dernier.

Le second avantage, déjà mentionné, est la possibilité d'encapsuler des applications, qui sont à la base non sécurisées (comme *telnet*, *pop*...), ce qui permet de leur faire bénéficier de toutes les garanties du protocole *SSH*.

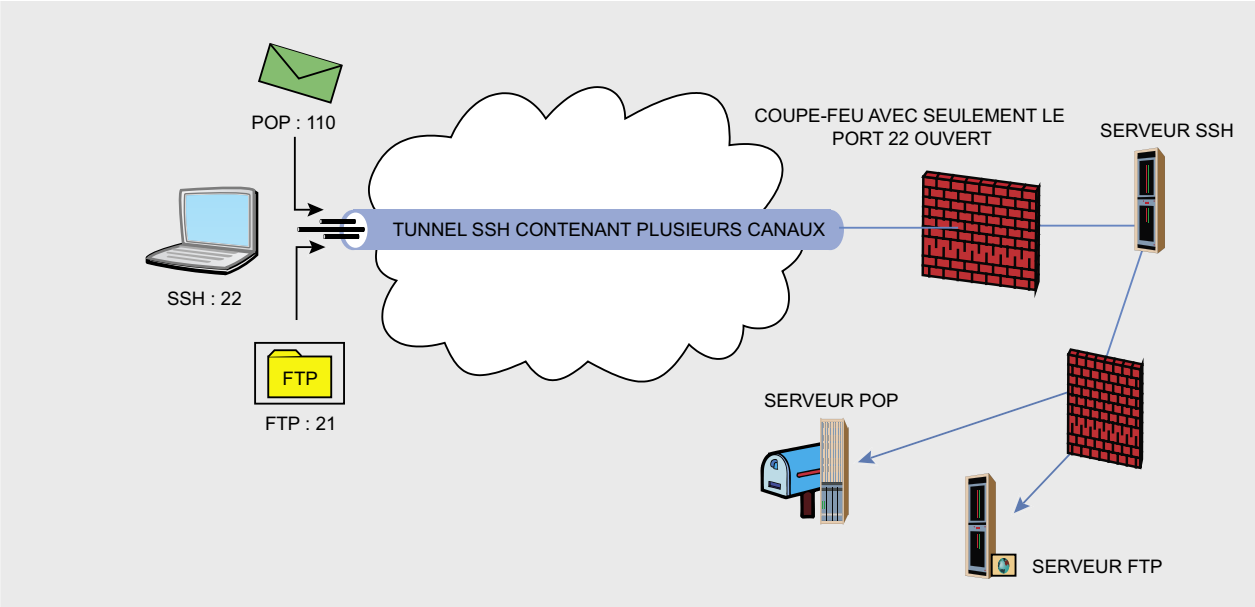

Figure 14. *Forwarding SSH*

Sites recommandés >>>

http://abcdelasecurite.free.fr

Abc de la sécurité informatique : portail dédié à la protection utilisateurs, la prévention et l'anonymat sur micro et réseaux.

http://www.commentcamarche.com

Dédié à l'informatique de A à Z. Il s'adresse aussi bien au débutants, qu'aux programmeurs à la recherche d'une information précise.

http://www.cccure.net

Le Portail Québécois de la Sécurité Des Systèmes. Site de nouvelles, forums, téléchargements, liens sécurité, et conseils.

http://www.virtuelnet.net

Construit sur des serveurs FreeBSD et Linux Infor-mations sur UNIX, Linux, Windows, les réseaux – Hébergement libre sur serveurs libres.

http://corporatehackers.com

Corporate Hackers fournit services et solutions dans le domaine de la sécurité des nouvelles technologies.

http://www.smtechnologie.com

Actualités, astuces, optimisation, personnalisation Windows, sécurité, visuels et téléchargement de logiciels, jeux et captures.

http://www.securiteinfo.com

SecuriteInfo.com, un des sites web leaders de la sécurité informatique francophone, propose ses services aux professionnels depuis 2004.

http://www.puissance-pc.net

Toute l'actualité informatique, les dernieres adresses, et beaucoup d'autres fonctionnalités, tout ceci géré par une équipe dynamique.

http://www.ze-linux.org

Crée par Terence DEWAELE, Ze-Linux a pour but d'aider la communauté française à utiliser GNU/Linux : Forums, Liste, News.

http://www.virustraq.com

VirusTraQ.com est un observatoire francophone sur les virus informatiques, le spam et les applications indésirables (spyware/adware).

http://www.vulnerabilite.com

Vulnerabilite.com est le portail francophone dédié à la sécurité des systemes d'information pour les DSI, RSSI et décideurs informatique.

Vous souhaitez vous aussi recommander votre site ?

Contactez-nous : *fr@hakin9.org*
Notre site : *http://www.hakin9.org/*

Implémentation et utilisation avancée de SSH dans un environnement Windows

Cette partie de l'article explique comment mettre en place un serveur SSH sous Windows.

Cette partie s'adresse donc aux administrateurs ne voulant pas s'embarrasser à configurer un serveur SSH sous Linux mais plutôt trouver une alternative pour garder leur bon vieux Windows. Il existe différentes solutions pour mettre en œuvre un serveur SSH sous Windows. L'une d'entre elles est Copssh, que l'on peut trouver à l'adresse suivante : *http://www.itefix.no* (une alternative est Cygwin). Son installation est très simple, il vous suffit de lancer l'exécutable téléchargé sur le site et de suivre les indications.

Une fois l'installation terminée, vous devez configurer un utilisateur apte à se connecter au serveur (par exemple *login*/*mdp*). Pour cela rendez-vous vers dans le répertoire d'installation du programme puis exécutez activate a *user.* Dans la nouvelle fenêtre, vous serez en mesure de configurer un nouvel utilisateur. Une fois cela effectué, il sera alors possible à cet utilisateur de se connecter au serveur.

Il existe là aussi différents clients SSH sous Windows. Le plus connu et le plus utilisé est sans doute putty. Maintenant que le serveur est installé et un utilisateur configuré, il ne vous reste plus qu'à ouvrir putty. Une fois lancé, tapez l'adresse du serveur SSH comme ci-dessous et enfin cliquez sur *open* (Figure 15).

Si votre serveur est bien configuré (et port 22 ouvert), vous arriverez à une fenêtre similaire à celle ci-dessous. Rentrez alors votre login et mot de passe (configuré au préalable sur le serveur SSH) : Voilà vous êtes connecté en SSH sur votre serveur.

Utilisation avancée de putty

Dans cette partie, vous apprendrez à configurer putty de manière à créer un tunnel et y faire transiter d'autres protocoles. Nous allons prendre pour exemple le protocole *Remote Desktop Protocol* permettant de prendre la main à distance sur un serveur Windows (*TSE*).

Dans un premier temps, nous allons nous assurer que seul le port 22 est ouvert sur le serveur. Ensuite, nous allons configurer putty. Pour cela, rendez-vous dans le menu *SSH* (volet de gauche) puis *Tunnels* : configurez ensuite votre tunnel comme suit (Figure 16). Remarquez qu'en port source il faut indiquer un port aléatoire non connu. D'autre part, en destination il

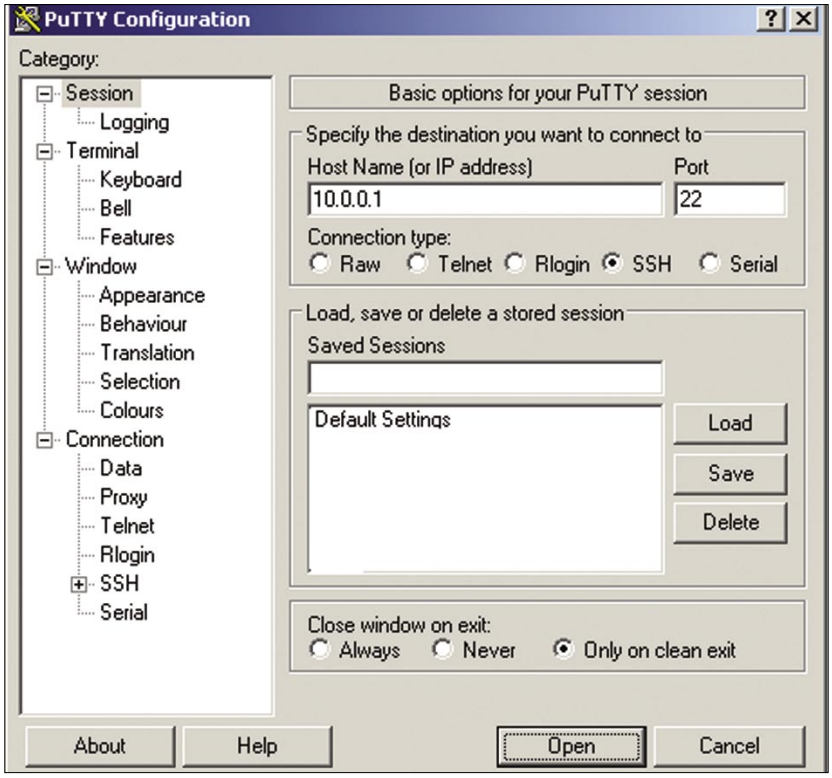

Figure 15. *Écran de configuration de Putty*

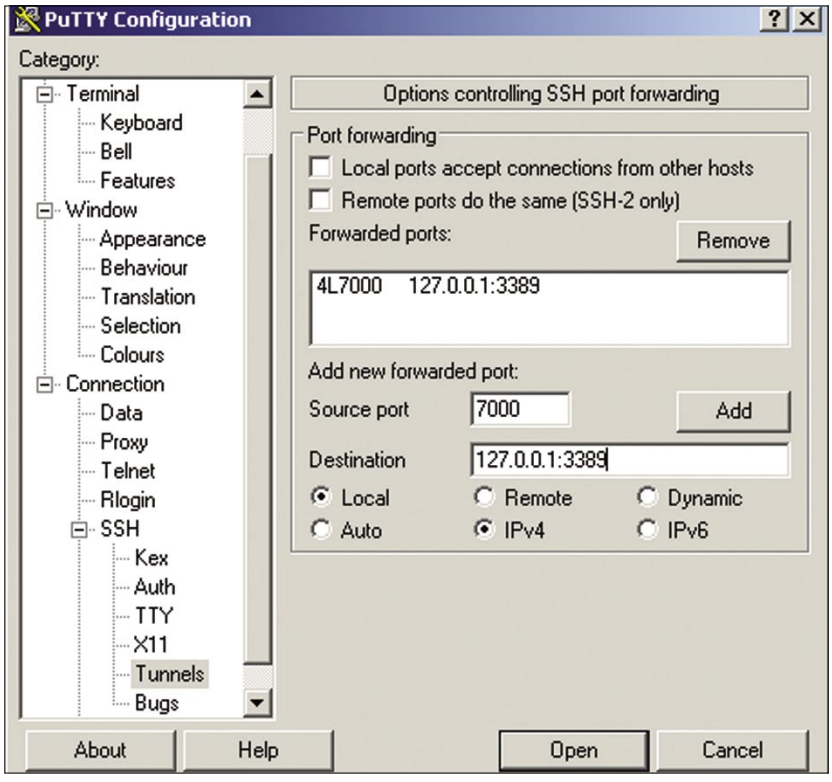

Figure 16. *Configuration tunnel SSH*

faut indiquer localhost suivi du port du service. Cela signifie que l'on établit un tunnel dans un premier temps puis que l'on se connecte au serveur *TSE* (donc lui-même) dans un second temps.

Revenez ensuite au menu principal, indiquez votre serveur *SSH* puis sauvegardez la configuration. Cliquez ensuite sur open puis renseignez votre login et mot de passe.

Maintenant que vous avez obtenu le *shell*, ouvrez votre client de connexion au bureau à distance et en guise d'adresse IP : *127.0.0.1:7000*.

Comme vous pouvez le deviner, le protocole RDP (utilisé pour le bureau à distance) est encapsulé et transite dans le tunnel sécurisé SSH !

Sécurisation d'un serveur SSH : génération de clés et utilisation de Pageant

Comme nous l'avons vu plus haut dans cet article, le protocole *SSH v2* supporte différents types d'authentification : par hôte, par mot de passe ou par clé publique. L'authentification par clé publique reste la plus fiable (en attendant la normalisation au niveau de *l'IETF* pour l'authentification basée sur les certificats...) c'est pourquoi nous allons aborder la mise en place de ce type d'authentification.

Cette méthode d'authentification repose, comme vous l'aurez certainement compris, sur la cryptographie asymétrique (*RSA* ou *DSA*) et nécessite donc que le serveur stocke la clé publique du client. Pour cela, le client doit d'une manière sûre transmettre cette clé au serveur *transfert sftp par exemple* ... Voici les différentes étapes pour sécuriser l'authentification par *SSH*. Dans un premier temps, côté client, il faut générer un couple de clés avec puttygen : lancer puttygen, sélectionner le type de clé à générer (exemple *SSH-2 DSA* et *2048*) puis cliquer sur generate. Rentrez alors une passphrase puis sauvez votre clé privée dans un endroit sûr.

Maintenant sélectionnez votre clé publique puis faites un *copier* (Figure 17). Il faut maintenant modifier sur le serveur le fichier *./ssh/authorized_keys* afin d'y coller votre clé publique.

Pour cela, vous pouvez utiliser SCP permettant de faire des transferts sftp avec une interface graphique rappelant les logiciels clients FTP qui se trouvent sous Windows (Figure 18). Maintenant, allez dans le dossier où vous avez sauvegardé votre clé privée et double cliquez dessus. Renseignez le passphrase puis faites un clic droit sur l'icône pageant qui se situe maintenant dans la barre des tâches. Allez dans saved sessions et sélectionnez la session que vous voulez démarrer (si vous avez auparavant sauvegardé une session putty). Vous n'avez alors plus que le login à renseigner et vous voilà authentifié avec votre couple de clés.

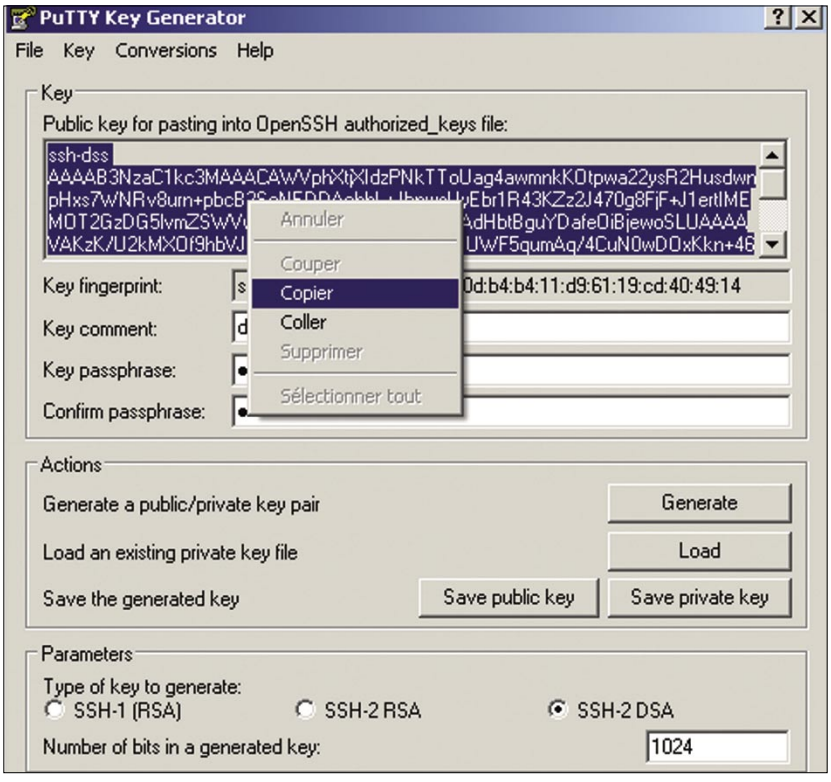

Figure 17. *Géneration de clés avec Puttygen*

Implémentation d'un serveur SSH Linux

Il est nécessaire d'etre en root pour effectuer toutes insallations concernant l'administration du systeme,on peut se logger: `$ su` Pour l'installation d'openssh-server, on peut donc soit effectuer la commande `sudo apt-get install` soit en root entrer directement `apt-get install openssh-server`

Nous allons voir ici les commandes et les configurations sous Linux. *OpenSSH*. Sous debian (ubuntu) :

```
aptitude install openssh-server
```

si la syntaxe n'est pas bonne il faut installer *aptitude* :

```
apt-get install aptitude
```

puis reprendre la première commande. Pour les utilisateurs d'ubuntu il faut faire la même syntaxe précédée de `sudo` pour installer en tant qu'administrateur. Sous redhat (fedora) :

```
yum install openssh-server
```

Il faut permettre seulement la version 2 à l'installation... (Installation par défaut).

Client

Sous debian (ubuntu) :

```
aptitude install openssh-client
```

si la syntaxe n'est pas bonne il faut installer aptitude :

```
apt-get install aptitude
```

Puis recommencer la première commande. Pour les utilisateurs d'ubuntu il faire la même syntaxe précédé de `sudo` pour installer en tant qu'administrateur.Sous redhat (fedora):

```
yum install openssh-client
```

Connexion

Nous partons du principe que vous avez crée un compte utilisateur florian et que vous l'ayez autorisé comme dans le fichié de configuration *sshd_config* qui se trouve plus bas. Se connecter au serveur SSH :

```
# ssh -p 53951 florian@ip_serveur_ssh
```

ou

```
# ssh -p 53951 -l florian ip_serveur_ssh
```

Pour avoir de l'aide on peut afficher le manuel grâce à la commande :

```
# man ssh
```

Configuration pour une utilisation avancée, plus sécurisée

Configuration du fichier SSHd_config

Ouvrir un shell en super utilisateur : `$ su -l`. Tapez votre mot de pass root. Nous allons éditer le fichier de configuration SSH :

```
# vi /etc/ssh/sshd_config
```

s'il ne se trouve pas ici tapez ceci pour le trouver :

```
# find / -name sshd_config
```

Dans ce fichier vous trouverez une multitude de choses :

```
Port 53951, Protocol 2, HostKey /etc/
      ssh/ssh_host_rsa_key, HostKey
      /etc/ssh/ssh_host_dsa_key,
      UsePrivilegeSeparation yes,
      KeyRegenerationInterval 3600,
      ServerKeyBits 768,
      SyslogFacility AUTH,
      LogLevel INFO, LoginGraceTime 20,
      PermitRootLogin no,
      StrictModes yes,
      AllowUsers florian,
      RSAAuthentication yes,
      PubkeyAuthentication yes,
      AuthorizedKeysFile /etc/
      ssh/UserKeys/authorized_keys.%u,
      IgnoreRhosts yes,
      PermitEmptyPasswords no,
      HostBasedAuthentication yes,
      PasswordAuthentication yes,
      X11Forwarding no, PrintMotd no,
      PrintLastLog no, KeepAlive yes,
      ListenAddress xx.xx.xx.xx,
      UsePAM yes.
```

J'ai choisi un port assez élevé (*53951*) peu commun évitant de ce fait les attaques sur le port classique *22*... Vous pouvez aussi modifier ces ports en éditant le fichier */etc/services*, en cherchant les lignes *ssh 22/tcp* et *ssh 22udp* et en modifiant le *22* par le nouveau port.

Si vous êtes derrière un proxy ou un pare-feu, il est préférable de laisser le port *22* car il sera dans la plupart des cas ouvert à l'inverse du port indiqué ci-dessus.

La fonction `AllowUsers` permet d'autoriser que les utilisateurs donnés. La fonction `ListenAddress` permet de spécifier l'adresse ip à partir de laquelle nous pourrons nous connecter.

Il est fortement conseillé de l'utiliser quand on possède une ip fixe. Pour avoir de l'aide pour les autres fonctions, on peut afficher le manuel grâce à la commande :

```
# man ssh_config
```

Pour que les modifications soient prises en compte, il faut redémarrer le service SSH `# /etc/init.d/sshd restart` ou `# service sshd restart`.

Génération de clés sur le client

Nous utiliserons des clés de cryptage asymétrique pour l'authentification. Qu'est-ce que le cryptage asymétrique? L'identification par cryptage asymétrique met en place 3 fichiers

- *~/.ssh/id_rsa* Clé privé qui permet de décrypter l'information et de s'identifier *Passphrase*,
- *~/.ssh/authorized_keys* Liste des clés publiques autorisées à accéder au compte de l'utilisateur,
- *~/.ssh/id_rsa.pub* Clé publique qui permet de crypter.

En bref, il sera possible de s'authentifier une bonne fois pour toutes à l'aide

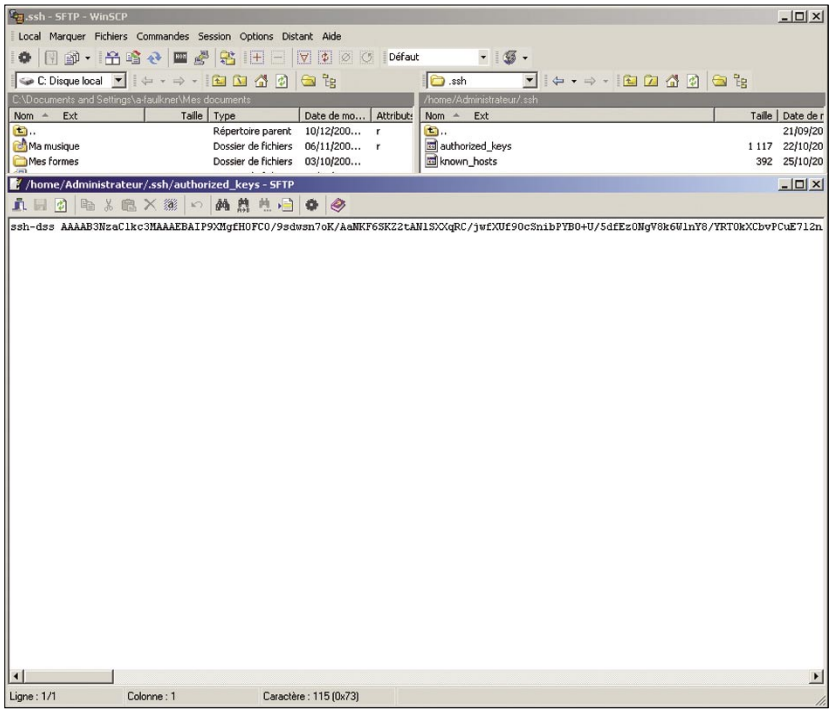

Figure 18. *Copie de la clé publique du client vers le serveur*

d'un agent qui reprend les trois fichiers ci-dessus petit logiciel interne. Utiliser *ssh-keygen* :

```
# ssh-keygen -t rsa
```

Étant loggé en utilisateur *florian* par défaut la clé se situera ici : */home/florian/.ssh/id_rsa*. Il vous sera demandé ensuite une *Passphrase* (une sorte de mot de pass). Je vous conseille d'utiliser beaucoup de caractères différents... Exemple :

```
je/suis\du-04/02/1988.et:hakin9=ca+rox
```

Retenez bien votre passphrase juste le temps de configurer l'agent. À la fin vous devez obtenur quelque chose comme :

```
the key fingerprint is :
ab:cd:ef:1a:c5:ab:cd:ef:7a:a6:ab:cd:
      ef:1d:c2:9b florian@localhost
```

Voir ici : `# man ssh-keygen`

Dans un shell tapez :

```
# ls -l /home/florian/.ssh/
```

Vous devriez voir :

- La clé *Privée* : *id_rsa* avec les droits *600*,
- La clé *Publique* : *id_rsa.pub* avec les droits *644*.

Maintenant nous allons transférer la clé publique sur le serveur *SSH* :

```
# ssh-copy-id -i /home/florian/.ssh
      /id_rsa.pub florian@ip_serveur_ssh
```

Un fichier *~/.ssh/authorized_keys* est créé sur le serveur *ssh*. Connectez-vous maintenant :

```
# ssh -p 53951 florian@ip_serveur_ssh
```

Entrez votre *PassPhrase* :

```
je/suis\du-04/02/1988.et:hakin9=ca+rox
root@ip_serveur_ssh:~$
```

vous voilà sur le serveur en connexion cryptée. Tapez : `logout`. Utilisation de ssh-agent. Tapez :

```
# ssh-agent screen
# ssh-add
```

Entrez votre *PassPhrase* :

```
je/suis\du-04/02/1988.et:hakin9=ca+rox
```

Voilà vous pouvez vous connecter maintenant en laissant l'agent s'occuper de tout.

```
# ssh -p 53951 florian@ip_serveur_ssh
```

Et vous voilà sur le server. Voir ici :

```
# man ssh-agent
```

Secure Copy

Nous allons procéder à un transfert de document vers le client avec *Secure Copy*

```
# scp -P 53951 -r florian@ip_serveur_
      ssh:/source/fichier destination
```

Exemple : je veux copier le fichier */root/Desktop/bonjour.txt* du server *ssh*

```
vers /home/florian/
```

la syntaxe sera :

```
# scp -P 53951 -r florian@ip_serveur_
      ssh:/root/Desktop/bonjour.txt
      /home/florian/
```

Copier un document du client vers le server ssh avec *Secure Copy* :

```
# scp -P 53951 -r /source/fichier
      florian@ip_serveur_ssh:destination
```

Ajoutez l'option `-p` à `scp -pr` pour préserver les droits et dates du fichier. Ajoutez l'option `-P` pour spécifier le port.

Attaques et Sécurisation sur le protocole SSH

Nous allons voir ici différentes possibilités d'intrusion. En effet, il existe certaines vulnérabilités ayant des conséquences variées. Il est à noter qu'il ne sera pas abordé dans cette partie l'attaque de l'homme du milieu, celle-ci étant traitée dans l'article suivant concernant également le *Port Forwarding*.

Attaque par Déni de Service – Denial of Service

Une attaque par déni de service consiste à mettre *hors service* une application voire même paralyser dans le pire des cas l'ordinateur. Le but de ce type d'attaque est d'empêcher la communication entre des clients et le serveur.

Comme nous l'avons vu, *SSH* s'appuie sur *TCP/IP*. Il hérite ainsi des faiblesses de la pile *TCP/IP* et est donc par conséquent vulnérable aux attaques par déni de service. IP ne permet pas de se protéger contre les injections de faux paquets. *SSH*, quant à lui, ne fournit aucune protection contre les fermetures de sessions *TCP*.

Cela permettrait donc à un individu d'injecter des paquets de manière à fermer les connexions *TCP* actives (*TCP* avec le flag *RST*).

Cette attaque profite des faiblesses du modèle TCP/IP et touche donc indirectement le protocole *SSH*. C'est ni plus ni moins qu'une attaque classique par déni de service.

Attaque par Brute Force

L'attaque de type *brute force* (également appelée attaque par force brute) consiste à essayer une authentification, de manière incrémentale, avec toutes les combinaisons de lettres/chiffres/caractères spéciaux pour trouver le mot de passe.

Cette méthode peut être combinée avec l'attaque par dictionnaire.

Ce type d'attaque est identique à celle concernant le protocole *FTP* par exemple (ou tous protocoles d'authentification traditionnelle basée sur le couple login-mot de passe). Différents logiciels permettent de réaliser cette attaque. Les plus utilisés sont : THC Hydra et Medusa. Il n'est pas inutile de préciser que ce genre d'attaque peut s'avérer très long, c'est pourquoi il est important de recueillir,

auparavant, un maximum d'informations sur le serveur et l'entreprise afin de générer un dictionnaire sur mesure... Il existe différentes mesures à prendre pour éviter ce type d'attaque. Tout d'abord, choisir un login autre que le classique *administrateur* ou *admin* !

Ensuite, il est impératif de choisir un mot de passe d'au moins 10 caractères alternants Lettres, Majuscules, Minuscules, Chiffres et caractères spéciaux et ne répondant à aucune logique : par exemple le mot de passe *Andrew1985!* n'est pas un bon mot de passe (mon prénom étant *andrew* et mon année de naissance *1985*). Il faut également faire attention à mettre un mot de passe différent de celui des autres serveurs.

Une autre ruse est de changer le numéro de port d'écoute du serveur *SSH* (par défaut *22*).

Pour cela, ouvrir le fichier */etc/ssh/sshd_config* et ajouter ou modifier la ligne *Port* comme suit: `Port 56789`.

Enfin, certains outils ou règles peuvent être instaurés afin de limiter ces attaques. Il est possible, par exemple, d'utiliser le logiciel *fail2ban* pour analyser automatiquement les logs de tentatives de connexion et bannir une adresse IP après un certain nombre de tentatives d'authentification infructueuses. Dans le cas du protocole SSH il surveille */var/log/auth.log* et lance des commandes *iptables* pour bannir les IP. Installation :

```
#sudo aptitude install fail2ban
```

Pour voir les actions du programme et le fichier de configuration :

```
#sudo cat /var/log/fail2ban.log
#cat /etc/fail2ban/jail.conf
[DEFAULT]
maxfailures = 3
bantime = 900
findtime = 600s
```

NB : Il contient déjà des lignes pour bloquer les attaques sur les serveurs *ftp* (*vsftpd*, *wuftpd*, *proftpd*...), *postfix*, *apache*... pour les activer il suffit de remplacer les lignes correspondantes avec `enabled = false` par `enabled = true`.

Par défaut, après 3 tentatives de connexion échouées sur le serveur SSH, l'adresse IP de l'attaquant sera bannie pour 10 minutes ce qui permet en général de parer une attaque par *brute force*.

La règle `IPTABLE` suivante aura pour conséquence de limiter le nombre de tentatives de connexion à 3 par heure :

```
-A INPUT -p tcp -m tcp -m limit -dport
      xxxx --limit 3/hour -j ACCEPT
```

Autres attaques possibles

Les attaques vues ci-dessus sont des attaques classiques et relativement faciles à mettre en œuvre.

Cependant, il existe plusieurs autres types d'attaques (plus exotiques) plus ou moins faciles à mettre en œuvre, mais qu'il est bon de connaître en tant qu'utilisateur de SSH.

Les premiers types d'attaques que nous vous exposerons relèvent de la logique. Admettons que l'authentification est réalisée via la clé publique du client (préalablement renseignée dans le fichier *authorized_key*).

Si l'on accède physiquement ou à distance à la machine dont la clé publique est autorisée, nous pourrions par exemple nous connecter au serveur à partir du poste client en nous basant sur sa clé privée (d'où l'importance de renseigner un login autre que administrateur mais surtout au niveau du poste client, choisir une *passphrase* complexe).

Nous pourrions également voler sa clé privée ou la modifier afin d'empêcher les futures communications du client vers le serveur.

Une autre méthode consisterait à élaborer un script qui, une fois exécuté sur le serveur avec le compte administrateur, permettrait d'ajouter la clé publique de l'attaquant dans le fichier *authorized_key*.

Si l'attaquant arrive à faire exécuter ce script par le serveur (libre à vous de réfléchir à la meilleure stratégie) il pourra alors se connecter au serveur sous réserve bien entendu de trouver le login approprié (il suffirait pour cela de modifier le script afin d'ajouter cet utilisateur apte à se connecter...).

Les autres attaques sont plus complexes à mettre en œuvre et peuvent donc paraître plus théoriques que pratiques. C'est pourquoi nous vous expliquerons le principe.

Nous vous inviterons ensuite à approfondir vos recherches si vous le souhaitez en vous indiquant quelques liens vers des sites Internet.

Attaque par analyse de trafic

SSH encapsule les données applicatives dans des paquets puis y ajoute un bourrage aléatoire.

Les données sont chiffrées par *SSH* puis envoyées en clair dans le champ `plaintextlength`.

Cela permet à un individu, qui écoute le trafic, de détecter la taille des données envoyées et de réaliser des attaques par analyse de trafic.

Comme pour le protocole *Telnet*, les caractères formant le mot de passe sont envoyés dans des paquets *IP* séparés Ainsi, en observant ces paquets, il serait possible de retrouver à la fois la taille du mot de passe, mais également de déterminer les caractères formant le mot de passe

En le couplant à un *brute force* cela accélèrerai grandement l'attaque et augmenterait les probabilités de réussite de l'attaque.

Cette attaque renvoi aux méthodes *Markoviennes* d'analyse de mot de passe (voir le lien dans la section sur Internet).

Exploitation du canal caché

Cette attaque est basée sur une faille connue comme étant un problème de sécurité récurrent dans les produits cryptologiques,

À propos des auteurs

Andrew Faulkner : Étudiant en sécurité des réseaux informatiques. Il poursuit actuellement des études en alternance avec HELIANTIS pour laquelle il réalise, entre autres, des prestations de sécurité auprès des clients. HELIANTIS, située à PAU, est une *Société de Services en Ingénierie Informatique* (SSII) et un *Fournisseur d'Accès Internet par Fibre Optique pour les Entreprises*. L'auteur peut être contacté à l'adresse : *andrewfaulkner64@gmail.com*.

Florian Blanc : Étudiant de 19 ans qui habite à Montauban. Il est actuellement en dernière année de formation de *Technicien Supèrieur* en *Réseaux Informatiques et Télécommunication d'Entreprises de niveau III* (Bac+2) + CCNA à l'Institut Informatique Sud Aveyron de Millau. Florian est auteur de la partie intitulée *Implémentation de SSH sous Linux* dans cet article. Il peut être contacté à l'adresse : *florian.blanc.adm@hotmail.fr*.

Sur Internet

- *http://openssh.com/* – Projet openssh,
- *http://www.ietf.org/rfc.html* – Voir les rfc 4250, 4251, 4252, 4253, 4254,
- *http://fail2ban.sourceforge.net/* – Site Officiel Projet fail2ban,
- *http://www.the-art-of-web.com/system/fail2ban/* – Fonctionnement de fail2ban,
- *http://denyhosts.sourceforge.net/* – Alternative à fail2ban,
- *http://www.csc.liv.ac.uk/~greg/sshdfilter* – Autre alternative à fail2ban,
- *http://www.openwall.com/advisories/OW-003-ssh-traffic-analysis/* – Analyse du trafic SSH,
- *http://www.om-conseil.com/sections.php?op=viewarticle&artid=15* – Exploitation du canal caché,
- *http://storm.net.nz/projects/7* – À méditer...

à savoir un canal caché également appelé subliminal que l'on peut exploiter.

La RFC 4251 concernant les détails sur l'architecture du protocole SSH, fait référence à cette faille de sécurité et je vous invite également à lire un article très intéressant sur cette faille (voir encadré *Sur Internet*).

Travaux pratiques sous Windows

Pour aller plus loin réalisez le TP suivant :

- *Matériel requis* : 3 ordinateurs sous Windows XP sp2 + un HUB.
- Premier ordinateur : pc-serveur : serveur *SSH* ; serveur *VNC* ; Serveur *Telnet* ; *Bureau* à distance activé ; *Port 22*, *5900*, *23*, *3389* ouverts | *192.168.0.1 /24*,
- Second ordinateur : pc-client : client putty ; client vnc | *192.168.0.2 /24*,
- Troisième ordinateur : pc-attaquant : *Wireshark* | *192.168.0.3 /24*.
- *TP n°1* : *Telnet* :
- Avec pc-attaquant lancez *Wireshark* et démarrez une capture en mode promiscuous,
- Avec pc-client connectez vous en *Telnet* sur pc-serveur. (rentrer `login+mdp`),
- Passez quelques commandes puis déconnectez-vous,
- Sur pc-attaquant arrêtez la capture puis retrouvez la communication entre pc-client et pc-serveur : voir que les données circulent en claire.
- *TP n°2* : *SSH*
- Réalisez le même *TP* avec une connexion *SSH*,
- Déterminez les informations transmises en clair,
- Conclure que *SSH* crypte les données.
- *TP n°3* : *Tunnel – Forwarding*
- Sur pc-serveur fermez tous les ports sauf le *22*,
- Réalisez un tunnel *SSH* et y encapsuler *telnet*,
- Vérifiez que l'on accède bien au serveur telnet mais que pc-attaquant ne voit plus les données.

Conclusion

Il a été expliqué, par le biais de cet article, le fonctionnement du protocole *SSH* et les différents algorithmes qu'il emploie dans le but d'assurer l'authentification, l'intégrité et la confidentialité des données.

Ce protocole sécurisé est une réelle évolution quand on le compare aux protocoles *Telnet* ou *Rlogin*.

En plus d'offrir un accès au *shell* de manière sécurisé, il est possible de sécuriser d'autres protocoles jugés moins sûrs (comme *telnet* ou *pop*...).

Pourquoi ne pas profiter de *SSH* pour faire du multi-tunneling applicatif et ainsi permettre de transmettre des données très confidentielles sur un réseau confidentiel ? D'autre part, le *tunneling* permet également de limiter le nombre de ports à ouvrir sur le garde-barrière et donc masquer les services qui tournent sur un serveur en interne ce qui est un point vraiment important dans l'établissement d'une architecture sécurisée.

Dans cet article, nous avons également abordé la mise en œuvre d'un serveur *SSH* sous deux environnements. Nous avons vu par exemple qu'il était possible d'installer un serveur *SSH* sous Windows même si cela pose certains problèmes au niveau des droits *POSIX*.

Enfin, une partie sur les différentes méthodes d'attaques et de protections nous a permi de voir qu'il était important que le serveur soit implémenté dans une architecture sécurisée.

L'utilisation de *SSH* doit constituer un élément cohérent d'une politique de sécurité globale et doit être mise en place en complément d'une politique d'accès et de filtrage, car malgré ses atouts, il n'est pas pour autant la panacée en matière de protocole sécurisé ... ●