Télécom Saint-Etienne

telecom
saint-étienne
école d'ingénieurs
nouvelles technologies

Projet TIPE - CITISE2

Cryptographie du Protocole SSH

Auteurs :
Thomas Cadegros
Sylvain Prost

Tuteur :
Florent Bernard

20 décembre 2023

Niveau de confidentialité : Ouvert

Table des matières

1 Introduction

Ces 200 dernières années, l'humanité s'est plus développée qu'elle ne l'a jamais fait dans toute son histoire. De la première révolution industrielle avec l'exploitation du charbon permettant l'avènement des machines à vapeurs, en passant par la deuxième avec l'exploitation du pétrole démocratisant le transport, la fin du XX^{e} siècle marque la dernière révolution[1]. Les nouvelles technologies d'information et de communication, permettant la transmission quasi-instantanée des données, a fait émerger Internet, une des plus grandes inventions de l'humanité.

Internet repose sur deux éléments : les infrastructures réseaux, et les protocoles de communication. Ces derniers définissent des règles précises concernant l'interaction entre des machines de l'infrastructure réseau. Les serveurs nous fournissent les services que nous utilisons au quotidien. Ces derniers étant dispersés sur la planète, il y a un besoin évident de pouvoir s'y connecter à distance, en tant qu'administrateur, non pas pour utiliser les services fournis, mais plutôt pour les paramétrer, échanger des fichiers, les mettre à jour, etc... Nous avons donc besoin des protocoles, et `telnet` et `rlogin` répondant aux besoins cités précédemment.

Mais la sécurité des communications est un enjeu majeur de l'ère du numérique, et les protocoles venant d'être cités ne répondent pas à ce critère. En effet, avec `telnet`, mis au point en 1969, toutes les informations sont transmises en clair sur le réseau. Lorsque que l'administrateur se connecte au serveur en saisissant identifiant et mot de passe, ces informations peuvent être saisies par un tiers qui espionne le réseau. Celui-ci peut alors prendre le contrôle du serveur. [2]

C'est pour ces problèmes de sécurité que le protocole SSH (Secure Shell) a été développé en 1995, puis normalisé dans sa version 2.0 en janvier 2006. SSH permet alors de répondre aux besoins cités précédemment, tout en assurant la sécurité de la communication au travers de trois points [3] :

- **L'authentification :** Pour que chaque machine soit sûre de communiquer avec celle qu'elle veut, et pas une autre machine pirate ;
- **La confidentialité des données :** Pour que personne ne puisse voir les données en clair ;
- **L'intégrité des données :** Pour être sûr que les données n'ont pas été altérées, accidentellement ou par un tiers.

La version 2 de SSH peut se décomposer en 3 couches [4] :

1. **Transport :** Assure l'authentification du serveur, la confidentialité et l'intégrité des données.
2. **Authentification :** Assure l'authentification du client. Est exécutée par dessus la couche Transport.
3. **Connexion :** Permet de multiplexer les services offerts par SSH (accès au shell, transfert de fichiers, tunneling, redirection de port) dans un seul canal sécurisé [3]. Est exécutée par dessus la couche Transport.

Dans ce rapport, nous ne nous intéresserons qu'aux deux premières couches, en analysant la cryptographie mise en jeu. Nous commencerons par étudier la phase de connexion, qui comporte l'analyse des trames et l'échange de clé, puis nous nous pencherons sur l'authentification du serveur et du client ainsi que sur le chiffrement des données.

2 | Connexion et échange de clé

2.1 Initialisation de la connexion

2.1.1 En-têtes des trames

Voyons tout d'abord le squelette d'une trame. Pour commencer, une trame envoyée ne contient pas juste une en-tête SSH et ses données. Elle contient d'abord une en-tête Ethernet, puis une en-tête IP ainsi qu'une en-tête TCP, correspondant respectivement à des protocoles ou des couches du modèle OSI liaison, réseau, transport et application. [5]

> *Le modèle OSI*, défini en 1978, est un ensemble de normes permettant à des équipements réseau de communiquer entre eux de manière fiable, au travers de différents protocoles. Le modèle est basé sur le fait qu'une communication peut se découper en plusieurs sous-étapes contenues dans différentes couches, au total 7, allant du plus bas niveau (couche physique) vers le plus haut niveau (couche application). Lors de l'émission d'une trame, on retrouve plusieurs protocoles, correspondantes aux différentes couches, encapsulées les unes dans les autres. Toutes les couches ne sont pas forcément représentée, comme c'est le cas ici. [6]

On retrouve en premier une trame du protocole Ethernet, qui ne contient que les adresses MAC source et destination, ainsi que le protocole de la couche suivante, ici IPv4. Cette partie sert à orienter une trame dans un réseau local et sera modifiée en cas de changement de réseau par un routeur. L'en-tête IPv4, contient entre autres les adresses IP des deux machines (client et serveur), ainsi que, là encore, le protocole de la couche suivante, TCP. Ce dernier, contient pour ce qui nous intéresse, les ports source et destination, ainsi que les numéros de séquences et d'acquittement, permettant une fiabilisation des échanges. [5]

> *Le protocole TCP* permet d'établir une connexion fiable avec une machine distante, avec des phases d'initialisation et de clôture de la connexion. Cela permet alors un mode « connecté ». Une connexion se déroule ainsi : le client envoie une séquence de synchronisation, auquel le serveur répond par une acceptation. Enfin, le client acquitte cette réponse. La fiabilité des échanges est assurée par un système d'accusé de réception, se basant sur les numéros de séquence et d'acquittement contenus dans les trames. [6]

Pour l'en tête du protocole SSH, on retrouve la taille du paquet, sur 8 octets, et la taille du bourrage sur 2 octets. Ce dernier permet d'avoir un nombre de bits compatible avec les algorithmes de chiffrement par blocs, qui chiffrent les données par blocs de même taille (cf. §3.2). On retrouve ensuite les données utiles, suivies du bourrage, constitué de bits générés aléatoirement. Enfin, on retrouve le MAC (Message Authentication Code), généré grâce à des fonctions de hachage. [3], [5]

> *Les fonctions de hachage* produisent une empreinte d'une taille déterminée, quelle que soit la chaîne passée en entrée. L'empreinte est la même pour une même chaîne passée en entrée. Le MAC est produit à partir d'un morceau des données utiles envoyées, passées dans une fonction de hachage. Celui qui reçoit les données peut alors prendre le même morceau des données utiles et le hacher. Le résultat doit alors être le même que le MAC reçu. Si ce n'est pas le cas, cela signifie que les données ont été altérées. [3]

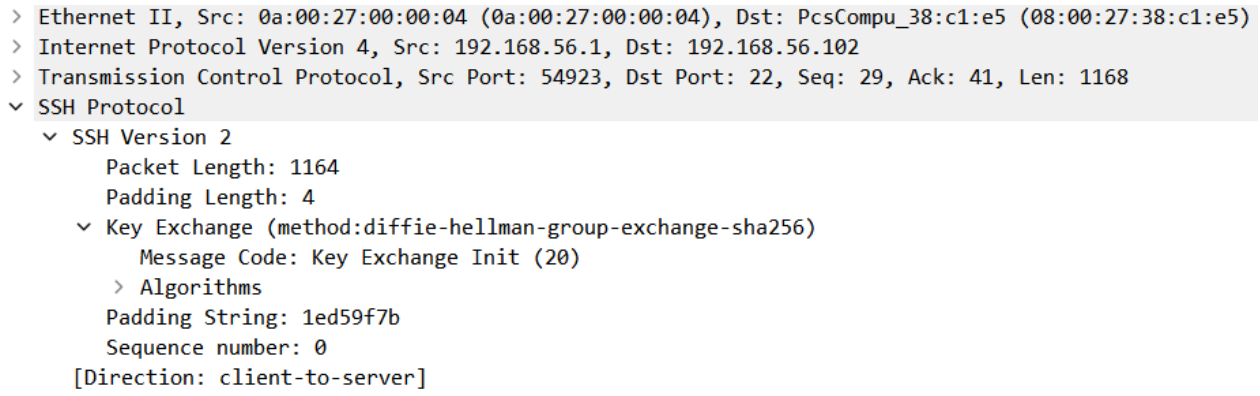

FIGURE 2.1 – Une trame SSH repose sur d'autres protocoles. Source : [5]

On reconnaît ici les différentes couches énumérées précédemment. « Key Exchange » représente ici les données utiles de cette trame SSH.

2.1.2 Hand-Shake

Nous allons maintenant détailler les échanges lors de la phase de « Hand Shake », soit la phase d'établissement de la connexion SSH. L'initialisation de cette connexion fait appel à la couche transport, et comprend [5] :

1. L'échange du numéro de version du protocole
2. L'initialisation de l'échange de clé
3. L'échange de clé Diffie-Hellman et l'authentification du serveur
4. La validation des algorithmes utilisés

Ci-dessous les trames échangées correspondante :

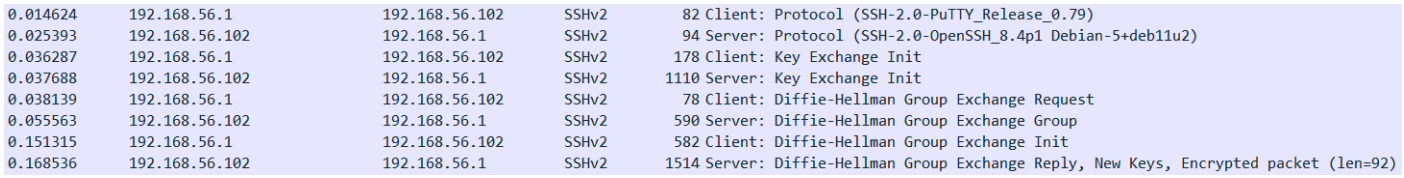

FIGURE 2.2 – Capture des premières trames non chiffrées avec Wireshark. Source : [5]

Échange du numéro de version du protocole La première trame SSH envoyée par chacune des deux machines ne comporte que le numéro de version du protocole avec la version du logiciel utilisant le protocole. Cette chaîne constitue la chaîne d'identification de chacune des deux parties et sera réutilisée par la suite (cf. §3.1.1.). L'écriture est de la forme : `SSH-versionprotocole-versionlogiciel commentaire CR LF`, où CR et LF représentent respectivement les caractères ASCII retour chariot et passage à la ligne.
Si les deux machines utilisent la version 2 de SSH, tout se passe bien. Le cas échéant, si le client utilise une version de SSH 1.X, le serveur peut parfois prendre en charge une compatibilité avec les anciennes versions. Dans ce cas, il rétrograde la version de son protocole afin de pouvoir communiquer avec le client.
Si le serveur ne prend pas en charge cette compatibilité, la connexion échoue. Si c'est le serveur qui utilise une version SSH 1.X, la connexion échoue également, et le client est invité à se reconnecter avec la même version que celle du serveur.[7]

Initialisation de l'échange de clé Cette étape, dont l'intitulé des trames est `SSH_MSG_KEXINIT` est cruciale à la bonne communication client-serveur, puisque chacun s'échange les algorithmes qu'il utilise pour les différentes fonctions nécessaires. Ces dernières sont la méthode d'échange de clé, l'authentification du serveur, le chiffrement, le MAC, et la compression. Ces algorithmes sont classés par catégorie, par ordre de préférence. Le client et le serveur peuvent alors deviner quel algorithme choisir. On retrouve aussi un Cookie généré aléatoirement par le serveur sur 32 octets, que le client lui renvoie, permettant d'éviter les attaques par déni de service (inonder un serveur de requête pour qu'il ne puisse plus fonctionner). [3], [5]

Si les deux machines ont le même algorithme d'échange de clé préféré, alors celui-ci est utilisé. Sinon, les machines regardent dans la liste du client chaque algorithme, dans l'ordre. Celui choisi sera alors le premier qui figure aussi dans la liste des algorithmes supportés par le serveur. Si aucun algorithme d'échange de clé n'est trouvé, la connexion échoue et les deux machines doivent être déconnectées. L'algorithme d'authentification du serveur est choisi de la même manière. Cette authentification sera détaillée plus tard (cf. §3.1.1). [7]

Les algorithmes suivants nécessitant une opération différente à l'envoi et à la réception de données, chacune étant l'inverse de l'autre (on peut faire l'analogie avec une bijection réciproque en mathématiques). On retrouve deux listes pour ces fonctions. Ainsi, l'algorithme de chiffrement possède une liste pour le sens client $\rightarrow$ serveur, et une pour le sens inverse. L'algorithme choisit d'être le premier dans les deux listes du client, et doit être dans les deux listes d'algorithme du serveur. Si aucun algorithme ne répond à ces critères, la connexion échoue et les deux machines doivent se déconnecter. Il en est de même pour les algorithmes de MAC et de chiffrement des données. [7]

L'échange de clé nécessite une description plus approfondie, la sous-partie suivante y est donc consacrée.

2.2 L'échange de clé

2.2.1 Définitions

La sécurité des algorithmes de cryptographie ne repose pas contrairement à la pensée populaire sur le secret du mécanisme, mais au contraire sur sa désignation "open source". Il s'agit du principe de Kerchoffs[8]. Un système dit open source rend public son fonctionnement. Cela permet à la communauté mondiale de pouvoir tester la sécurité de ses algorithmes. Ainsi, la sécurité repose sur un autre élément essentiel : les clés de cryptage.

Cryptographie symétrique Les algorithmes cryptographiques symétriques permettent de chiffrer des informations en utilisant une unique clé, qui permet à la fois de chiffrer et déchiffrer. Un problème se pose alors : l'échange de la clé entre deux interlocuteurs, de manière sécurisée.
Le chiffrement symétrique repose sur des algorithmes faisant appel à des opérations arithmétiques basiques précablées, dans des processeurs possédant une architecture RISC (Reduced Instruction Set Computer)[8], c'est à dire les processeurs présents dans la majorité des systèmes. Ils permettent ainsi de crypter des messages plus rapidement que les algorithmes cryptographiques asymétriques.

Cryptographie asymétrique Contrairement à la cryptographie symétrique, la cryptographie asymétrique utilise quatre clés. Chaque entité en possède deux : une clé publique, connue de tous, et une clé privée qu'elle garde secrète. L'avantage majeur de la cryptographie asymétrique est qu'elle est à double face. En prenant les personnages Alice et Bob, représentant chacun une machine, nous avons [9] :

Chiffrement des messages Pour que Alice envoie un message à Bob, elle utilise la clé publique de Bob pour chiffrer ce message. Bob, et uniquement lui, est en mesure de déchiffrer le message, grâce à sa clé privée. Il n'y a donc pas de problème de transmissio de clé.

Authentification d'une machine On peut appliquer l'opération inverse pour qu'une machine prouve son authenticité, c'est-à-dire que personne ne puisse se faire passer pour elle. Pour qu'Alice puisse

prouver à Bob, généralement, Bob devra envoyer un message quelconque à Alice. Cette dernière chiffrera ce message (on dira plutôt signer un message), avec sa clé privée qu'elle envoie à Bob. Vu qu'il possède la clé publique d'Alice, il pourra alors déchiffrer le message. Si le message reçu est identique à celui envoyé, alors Bob peut être certain que le message provient de d'Alice, car seule Alice est propriétaire de sa clé privée.

Ces algorithmes répondent au principe de sécurité sémantique[10]. Considérons un attaquant interceptant un message crypté : pour que l'algorithme réponde à ce principe, il doit lui être impossible d'extraire une information du message en clair en un temps humainement raisonnable. Pour crypter les informations, la clé publique peut être transmise à l'interlocuteur en clair sans remettre en cause la sécurité, tandis que la clé privée devra rester secrète pour garantir la sécurité de la communication.

2.2.2 Mise en place d'un chiffrement symétrique

L'utilisation d'un algorithme symétrique dans un protocole de communication semble ainsi plus adaptée, cependant, les algorithmes asymétriques leurs sont complémentaires. Leurs clés publiques permettent de chiffrer une information sans avoir partagé une clé commune à deux interlocuteurs.// Le protocole de chiffrement utilisé dans notre transmission SSH est le ChaCha20. Son algorithme cryptographique utilise des clés symétriques de 256 bits[11]. L'échange de ces clés entre le client et le serveur est réalisé par l'algorithme de Diffie et Hellman.

2.2.3 Algorithme de Diffie et Hellman

Le but de l'échange de Diffie-Hellman est d'obtenir un secret partagé K.

Notion de groupe

Commençons par définir la notion de groupe, il s'agit d'un ensemble muni d'une loi interne $\times$ telle que : (notons G un groupe)[12]

- $\times$ est associatif, soit $\forall x, y, z \in G\ :\ x \times (y \times z) = (x \times y) \times z$
- G contient un élément neutre e tel que $\forall x \in G\ x \times e = e \times x = x$
- Tout $x \in G$ possède un symétrique y tel que $x \times y = y \times x = e$

Un ensemble H est un sous groupe de G si et seulement si [13] :

- L'élément neutre de G est contenu dans H : $e \in H$
- Soit $x, y \in H$ et y^{-1} le symétrique de y alors $x \times y^{-1} \in H$

Considérons l'ensemble $\mathbb{Z}$ muni d'une loi multiplicative $\times$. La relation d'équivalence de $\mathbb{Z}$ entre deux nombres relatifs x, y est vraie si et seulement si l'entier relatif n divise $x - y$. Ainsi, le sous ensemble de $\mathbb{Z}$ noté $\mathbb{Z}/n\mathbb{Z}$ représente l'ensemble d'équivalence de x, soit les restes de la division euclidienne de x par n soit : $\{0, 1, ..., n-1\}$.[14]

Considérons l'ensemble $(\mathbb{Z}/p\mathbb{Z})\backslash\{0\}$, ainsi que la loi suivante avec x et y des éléments du groupe : $(x\ mod\ p) \times (y\ mod\ p) = xy\ mod\ p$. Rappelons le théorème de Bezout, soit $k, p \in \mathbb{Z}$ alors k et p sont premiers entre eux si et seulement si il existe un couple d'entiers relatifs (u, v) tel que $uk + vb = 1$. Ainsi, pour que chaque ensemble $(\mathbb{Z}/p\mathbb{Z}\backslash\{0\}, \times)$ possède un inverse : il faut que p soit un nombre premier.[14]

Un groupe est cyclique s'il existe un élément g du groupe tel que pour tout élément y du groupe, il existe $k \in \mathbb{Z}$ tel que $y = g^k$. En appliquant le théorème de Bezout, le groupe ayant pour générateur g est donc cyclique si et seulement si g et p sont premiers entre eux. Il s'agit d'un groupe fini.

Ainsi, nous disposons d'un sous groupe de $(\mathbb{Z}/p\mathbb{Z}\backslash\{0\}, \times)$ de générateur g noté $< g >= \{a^k, k \in \mathbb{Z}\}$, l'ordre de ce sous-groupe étant fini son cardinal correspond à l'ordre du groupe noté $o(g)$,[15] par définition

$o(g) := min\{m \in \mathbb{N}^* | g^m \ mod \, n = 1\}$.

Problème de Diffie et Hellman

L'algorithme de Diffie et Hellman repose sur des calculs par des puissances ainsi que par des opérations de congruence, le principe de sécurité sémantique repose sur le problème du logarithme discret.

Considérons un groupe cyclique $G =< g >$ d'ordre n de générateur g. Ainsi tout élément x de G peut s'écrire sous la forme $x = g^\alpha$ avec $1 \leq \alpha \leq n$ ainsi α est le logarithme discret de x en base g noté $log_g(x)$.

Considérons le nombre $g^{ab} \ mod \, n$, nous cherchons à retrouver ab que nous notons y, pour cela, utilisons l'algorithme "baby-step giant-step"[16], notons $h = g^y$, soit $m = \lceil \sqrt{n} \rceil$ le premier entier supérieur à $\sqrt{n}$:

Algorithm 1 : baby-step giant-step

Data : Valeurs de g et $m = \lceil \sqrt{n} \rceil$
Result : y
$i = 1$;
while $i \leq m$ **do**
  Placer $g^i \ mod \, n$ dans la table T ;
  i++ ;
end
$j = 1$;
while $j \leq m$ *ET* $h \times g^{j \times -m} \ mod \, n$ *n'est pas présent dans* T **do**
  j++ ;
end
Retourner $y = i + jm$;

On obtient $g^i = h \times g^{-jm}$ on a donc $y = i + jm$. Cet algorithme est le plus puissant connu à ce jour pour résoudre le problème du logarithme discret[17]. Il possédera un temps d'exécution de l'ordre de m, il met en avant la nécessité de posséder un entier g très grand, ceci dans le but de rendre impossible la résolution de cet algorithme dans un temps humainement concevable. L'Agence nationale de la sécurité des systèmes d'information imposait en 2020 une taille de minimum d'ordre de groupe de $3072bits$ pour les systèmes devant dépasser l'an 2030[18].

Échange d'un secret avec Diffie et Hellman

La méthode cryptographique de Diffie et Hellman est utilisée pour permettre à deux personnes de s'échanger un nombre secret sans le divulguer lors des échanges. Pour commencer l'échange, les deux interlocuteurs doivent se mettre d'accord sur un entier de très grande taille. Ce nombre peut être généré selon deux types de groupes :

1. Les corps finis, qui correspondent aux groupes cycliques. Dans ce cas, ils s'échangent le générateur du groupe et l'ordre du groupe.
2. Les courbes elliptiques, que nous n'étudierons pas en raison de la complexité des points mathématiques abordés.

Soit l'échange du secret entre deux personnes (Alice et Bob) :

Alice			Bob		
Secret	Calcul	public	public	Calcul	Secret
		p, g	p, g		
a					b
	$A = g^a \ mod \, p$	A	reçoit A		
		reçoit B	B	$B = g^a \ mod \, p$	
	$K = B^a \ mod \, p = g^{ab} \ mod \, p$			$K = A^b \ mod \, p = g^{ab} \ mod \, p$	

TABLE 2.1 – Algorithme de Diffie et Hellman

A la fin de cet algorithme, Alice et Bob détiennent le même secret $g^{ab} \ mod \, p$.
Supposons qu'un attaquant écoute les transmissions entre Alice et Bob, il se retrouve en possession de 4 éléments g, p, A, B. Pour pouvoir être en mesure de prendre connaissance de la clé secrète, il doit être capable de résoudre une des deux équations suivantes :

$$g^a \ mod \, p = A \ mod \, p \qquad OU \qquad g^b \ mod \, p = B \ mod \, p$$

Avec b et a les inconnus du problème. La résolution de ces équations fait intervenir le problème du logarithme discret, pour garantir la sécurité de la clé secrète, il est donc indispensable d'utiliser des nombres d'un ordre de grandeur très grand.

Diffie et Hellman appliqué au SSH

Dans notre étude, nous nous intéressons à la méthode d'échange de clé `diffie-hellman-group-exchange`, intégrée au SSH.

Le serveur et le client se mettent d'accord sur le groupe à utiliser. Pour ce faire, le serveur possède une liste de nombres premiers dits sûrs ; renouvelée fréquemment en fonction du réglage du serveur SSH, il choisit le nombre premier $p = 2q + 1$ tel que q soit sûr. Le nombre q est généré aléatoirement et p est calculé, cette opération est répétée jusqu'à ce que p soit premier[19]. L'ordre du sous groupe généré par le générateur g devra être compris entre des nombres s'écrivant entre 1024 et 8192 bits. De plus, l'ordre ne peut pas être un facteur d'un "petit" nombre premier, il devra être de q ou $p - 1$ étant donné que p et q sont premiers. Une fois que le serveur possède les nombres g et p correspondants aux différents critères, il les transmet par une seconde trame au serveur[20].

Le serveur et le client génèrent leurs nombres secrets a et b. Afin d'appartenir au sous-groupe choisi par le serveur, ces nombres doivent respecter le critère suivant : $1 < a, b < \frac{p-1}{2}$. Cette limite haute provient du fait que le plus petit ordre possible est q et $q = \frac{p-1}{2}$[20]. L'échange se passe ensuite comme décrit précédemment (cf. §2.2.3

3 | Authentification et chiffrement des messages

3.1 Authentification du serveur et du client

3.1.1 Authentification du serveur

L'authentification du serveur dans le protocole SSH se fait lors de l'échange de Diffie-Hellman (cf. §2.2.3). On reprend ici les valeurs de l'échange : $A = g^a \, modp$, $B = g^a \, modp$ et $K = A^b \, modp = B^a \, modp = g^{ab} \, modp$.

Client → Serveur Le client calcule A et l'envoie au serveur. Ce dernier va alors directement calculer B et K. Il va aussi calculer H, obtenu grâce à une fonction de hachage appliquée sur la concaténation :

- Des chaînes d'identification, celle du client que l'on notera VC et celle du serveur que l'on notera VS (cf. §2.1.2) ;
- Du message `SSH_MSG_KEXINIT`, celui envoyé par le client que l'on notera IC ainsi que celui envoyé par le serveur que l'on notera IS (cf. §2.1.2) ;
- De la clé publique du serveur, on la notera KS_{pub}
- De A, envoyé par le client
- De B, qu'il vient de calculer
- De K, qu'il vient de calculer

Il faut noter que ces valeurs sont connues du client et du serveur, une fois que ce dernier a envoyé sa trame d'échange Diffie-Hellman. Le serveur chiffre ensuite H avec sa clé privée, que l'on notera KS_{priv}, ce qui donne S. Il envoie enfin au client KS_{pub}, B, et S. [7]

Serveur → Client Le client reçoit ces informations et va vérifier que la clé d'hôte publique appartient bien au serveur. Pour cela, il va vérifier la correspondance sur une base de données locale, ou avec un tiers de confiance, les serveurs PKI. Si aucune correspondance n'est trouvée, le client en est averti et peut choisir : de se déconnecter ou de poursuivre la connexion quand même. Cette méthode reste déconseillée car elle ne permet pas d'assurer réellement l'authentification du serveur. Si la connexion se poursuit quand même, le client calcule à son tour K et H (H est calculé de la même manière que décrite ci-dessus). Il peut alors déchiffrer S grâce à la clé publique de l'hôte, et retrouver le H que le serveur a calculé. Ce dernier doit être identique à celui calculé par le client.[7]

> *Les PKI* (Public Key Infrastructure) sont des serveurs permettant à des clients d'obtenir la clé publique d'une machine de manière sécurisée, en étant certain que la clé donnée correspond bien à la machine voulue. La fiabilité de ces serveurs reposent sur une chaîne de confiance avec à la base un serveur que l'on sait « vraiment fiable » [21]

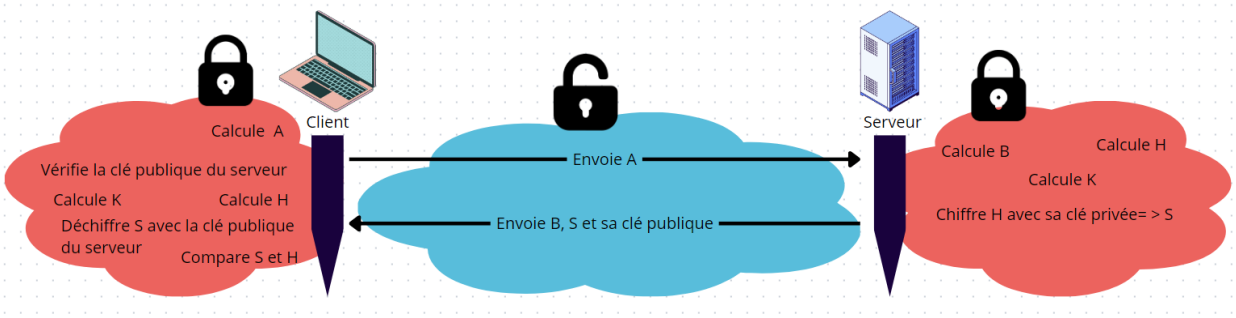

FIGURE 3.1 – Authentification du serveur. Source : [22]

Une fois l'échange de clé terminé, le message `SSH_MSG_NEWKEYS` est envoyé par les deux parties et signale qu'à partir de ce message, les algorithmes utilisés et les clés de chiffrement utilisées seront ceux qu'ils viennent de définir. [7]. C'est la fin de la hand-shake énoncée §2.1.2.

3.1.2 Authentification du client

Une fois l'hôte identifié, le client envoie une demande de service, c'est à dire une demande d'activation de la couche d'authentification ou de la couche connexion. La demande de ce dernier service est rejetée par le serveur si le client ne s'est pas authentifié avant, les machines se déconnectent alors. Le client doit donc d'abord s'authentifier [3]. Pour que ce dernier puisse le faire, 4 méthodes s'offrent à lui :

Méthode *none* Cette méthode ne requiert aucune authentification auprès du serveur, n'importe quel client possédant l'adresse IP et le nom d'utilisateur du serveur peut s'y connecter. Il est évidemment fortement recommandé de ne pas utiliser cette technique.[7]

Méthode *password* Le password est une chaîne de caractères connue du serveur. Lors de la connexion du client au serveur, cette chaîne lui est demandée. Elle est transmise en clair dans un paquet au serveur, cependant, le paquet reste crypté. Une fois le password récupéré par le serveur, il est vérifié selon les méthodes de l'OS installé sur le serveur et autorise la connexion au serveur.[7]

Méthode *publickey* Cette méthode est la plus courante et doit être obligatoirement activée, elle est la plus recommandée. Une *passphrase* est générée par le serveur et transmise en clair via les échanges cryptés au client. Cette *passphrase* sera cryptée par un chiffrement asymétrique, le client cryptant avec sa clé privée l'information. Elle est transmise désormais cryptée au serveur via les échanges cryptés. Le serveur possède les clés publiques des clients autorisés à se connecter. Il déchiffre donc l'information avec la clé publique du client. Si l'information décryptée et la *passphrase* concordent, le client est autorisé à se connecter. [7]

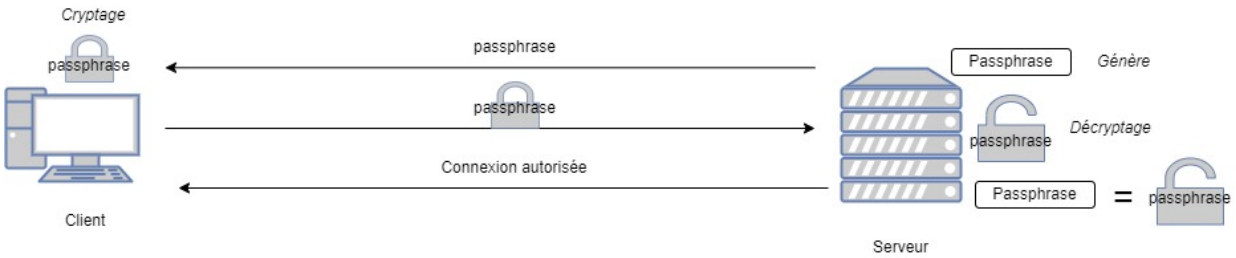

FIGURE 3.2 – Authentification du client auprès du serveur : Méthode **publikey** - Source : [23]

Méthode *hotbased* Cette méthode d'authentification est basée sur l'hébergeur du client, elle permet à l'ensemble des clients de l'hébergeur d'obtenir un accès distant au serveur, il est donc indispensable de prendre

des précautions lors de la mise en place de cette méthode qui pourrait donner des privilèges à n'importe quel utilisateur. Lorsque le client se connecte, le serveur vérifie que le nom d'hôte est présent dans son fichier *rhosts*, et vérifie si le programme demandé est autorisé en vérifiant si son port d'écoute est compris entre 1 et 1023. En effet, ces ports d'écoute sont dits sûrs puisque activables seulement par l'administrateur.[3]

3.1.3 Authentification par RSA

La méthode d'authentification par publickey du client ou celle du serveur nécessite une utilisation d'une méthode cryptographique asymétrique. Pour cela, nous pouvons utiliser le RSA. Commençons par introduire le cryptage RSA au sens global.

Groupe multiplicatif

Considérons le groupe multiplicatif $(\frac{\mathbb{Z}}{n\mathbb{Z}})^\times$ (cf. §2.2.3) soit l'ensemble des éléments inversibles du groupe $(\frac{\mathbb{Z}}{n\mathbb{Z}})$, considérons k un élément de $(\frac{\mathbb{Z}}{n\mathbb{Z}})^\times$, montrons qu'alors k et n sont premiers entre eux.

Si $k \in (\frac{\mathbb{Z}}{n\mathbb{Z}})^\times$, alors il existe un entier $l \in (\frac{\mathbb{Z}}{n\mathbb{Z}})^\times$ tel que l soit l'inverse de k donc $k \times l = 1$, il existe donc $m \in \mathbb{Z}$ tel que $kl-1 = mn \Leftrightarrow kl-mn = 1$, d'après le théorème de Bezout (cf. §2.2.3), n et k sont premiers.

Ainsi, le cardinal de $(\frac{\mathbb{Z}}{n\mathbb{Z}})^\times$ correspond au nombre d'éléments k tels que k et n soient premiers entre eux, soit la fonction d'Euler de n notée $\varphi(n)$.[24]

Génération de clés RSA

Prenons deux nombres premiers distincts p et q tels que $n = pq$, le cardinal du groupe $(\frac{\mathbb{Z}}{n\mathbb{Z}})^\times$ est donc $\varphi(n) = (p-1)(q-1)$ [25]. On génère un entier e premier avec $\varphi(n)$ et son inverse d,

$$d \times e = 1 \, mod \, (\varphi(n))$$

Nous pouvons utiliser l'algorithme d'Euclide étendu pour déterminer l'inverse modulaire de e noté d [25].

Algorithm 2 : Algorithme d'Euclide étendu pour trouver l'inverse

Data : Valeurs de x et $\varphi(n)$
Result : Inverse de $x \, mod \, \varphi(n)$
$a \leftarrow$ valeur de x ;
$b \leftarrow$ valeur de $\varphi(n)$;
$u1 \leftarrow 1$, $v1 \leftarrow 0$;
$u2 \leftarrow 0$, $v2 \leftarrow 1$;
while $b \neq 0$ **do**
  $quotient \leftarrow a \div b$;
  $reste \leftarrow a \mod b$;
  $u \leftarrow u1 - quotient \times u2$;
  $v \leftarrow v1 - quotient \times v2$;
  `// Mise à jour des variables pour la prochaine itération`
  $a \leftarrow b$;
  $b \leftarrow reste$;
  $u1 \leftarrow u2$;
  $v1 \leftarrow v2$;
  $u2 \leftarrow u$;
  $v2 \leftarrow v$;
end
if $a = 1$ **then**
  `// À la fin de la boucle,` $u1$ `est l'inverse de` $a \, mod \, \varphi(n)$
  $inverse \leftarrow u1 \mod b$;
  Retouner $inverse$;
end
else
  Retourner "L'inverse n'existe pas, car", a, "et", b, "ne sont pas premiers entre eux." ;
end

Nous obtenons deux couples $Pr = (e, n)$ $Pu = (e, d)$, les clés respectivement privées et publiques de notre système.

Fonctions de cryptage RSA

La fonction de cryptage de la cryptographie RSA est la suivante, avec x l'élément à chiffrer [25] :

$$F(x) = x^y \, mod \, n$$

y correspond à d ou e en fonction de la clé de cryptage utilisée. L'élément à chiffrer peut-être un nombre représentant des données comme du texte converti en binaire avec le langage ASCII. Montrons désormais que cette fonction de cryptage peut permettre de décrypter une information cryptée avec la première clé à l'aide de la seconde. Pour cela, démontrons que les fonctions suivantes sont l'inverse l'une de l'autre :

$$Fpu(x) = x^d \, mod \, n$$
$$Fpr(x) = x^e \, mod \, n$$

Prenons un élément x à crypter, en le cryptant avec la fonction $Fpu(x)$ nous obtenons $x^d mod n$. Désormais passons ce nombre dans la seconde fonction de cryptage, nous obtenons $x^{de} \, mod \, n$, montrons que ce nombre est égal à $c mod n$. Rappelons que $n = pq$, d'après le petit théorème de Fermat, $(c mod p)^{p-1} mod p = 1$ or e et d sont des inverses multiplicatifs modulo $\varphi(n)$. Soit pour un entier h, on peut écrire $ed = 1 + h(p-1)(q-1)$. Si $t \, mod \, p \neq 0$ alors :

$$\Leftrightarrow x^{ed} \, mod \, p = (x \, mod \, p)^{ed} \, mod \, p$$
$$\Leftrightarrow x^{ed} \, mod \, p = (x \, mod \, p)^{1+h(p-1)(q-1)} \, mod \, p$$

$$\Leftrightarrow x^{ed} \bmod p = ((x \bmod p).((x \bmod p)^{p-1} \bmod p)^{h(q-1)}) \bmod p$$
$$\Leftrightarrow x^{ed} \bmod p = (x \bmod p).(1^{h(q-1)} \bmod p)$$
$$\Leftrightarrow x^{ed} \bmod p = x \bmod p$$

Si $x \bmod p = 0$ alors $x^{ed} \bmod p = 0$. De même, on peut montrer Si $x \bmod q = 0$ alors $x^{ed} \bmod q = 0$ et si $x \bmod q \neq 0$ alors $x^{ed} \bmod q = x \bmod q$. De plus, si $x^{ed} \bmod p = x \bmod p$ et $x^{ed} \bmod q = x \bmod q$ alors $x^{ed} \bmod pq = x \bmod pq$, sachant que $n = pq$ on obtient :

$$x^{ed} \bmod n = x \bmod n$$

Les fonctions de cryptage par les clés publiques et privées sont donc des fonctions inverses, elles permettent de crypter ou décrypter l'information cryptée par une autre clé [25].

Problème de factorisation des grands entiers

La sécurité du chiffrement RSA, repose sur le principe de factorisation du modulo n. En effet, si un attaquant est en possession de n, et s'il est capable de le factoriser en deux nombres premiers p et q alors, en faisant les opérations du §3.1.3, il est en mesure d'obtenir les clés publique et privée.[26]

L'algorithme NFS est l'algorithme le plus rapide connu à ce jour pour résoudre ce problème [26]. Il repose sur le principe de la congruence carrée, en recherchant des paires d'entiers x et y telles que

$$x^2 = y^2 \bmod n$$

Ceci avec pour objectif de trouver un facteur de n en calculant $pgcd(x-1, n)$ et $pgcd(x+y, n)$. [27]. Les étapes de l'algorithme sont les suivantes :

1. **Sélection ploynomiale :** Pour commencer, nous devons choisir deux paires de polynômes irréductibles à coefficients entiers : f_0 et f_1 premiers entre eux et possédant une racine commune notée $m \bmod n$.[27]
2. **Collecte des relations :** Cette étape consiste à identifier deux entiers $Res(a-bx, f_0)$ et $Res(a+bx, f_1)$ des entiers résultants de chaque polynôme, dans un espace indexé par les entiers a et b. De sorte que leurs plus grands facteurs premiers soient plus petits à deux entiers fixés (deux bornes). Cette étape étant coûteuse en ressource, on peut se permettre d'utiliser de grandes bornes. [26]. Des relations obtenues, il en est déduit plusieurs sous ensembles.
3. **Algèbre linéaire** De ces relations et sous-ensembles est obtenu un système linéaire.[26]
4. **calculs finaux :** Des calculs finaux faisant appel à des techniques d'arithmétique multiprécision sur des entiers de très grandes tailles permettent d'obtenir les nombres p et q. [26]

Lors de ces étapes, la recherche des relations ainsi que l'algèbre linéaire sont très coûteuses en ressources. Par exemple, pour un modulo de $829bits$, la recherche de relation pour un processeur Intel Xeon 6130 (2.1 GHz) est de 794 années-cœurs [26] et la recherche totale des entiers p et q 953 années-cœurs [26]. La factorisation d'un modulo de 1 024 bits devrait être bien plus longue, avec une durée de 500 000 années-cœurs. [26].

Ceci met en évidence la nécessité d'utiliser des modulos de grandes tailles pour garantir la sécurité des échanges. Ceci est d'autant plus important que les clés RSA sont très peu renouvelées.

RSA et SSH

Ainsi, nous privilégierons pour garantir la sécurité de l'authentification une clé RSA-2048 composée d'un modulo de 2048 bits. Il est nécessaire que le client ait transmis au serveur en amont sa clé publique. Ceci peut se dérouler lors d'une précédente session, directement sur le serveur, ou via un service externe comme dans le cas de GitHub qui propose de rentrer sa clé RSA publique directement dans les paramètres de son compte. Dans le cas de l'authentification du serveur, la clé publique est transmise durant l'échange de clé de Diffie-Hellman (cf. §3.1.1).

3.2 Chiffrement des messages avec ChaCha20

Nous allons ici nous pencher sur l'algorithme de chiffrement `ChaCha20`. C'est l'algorithme de chiffrement le plus rapide, et c'est celui qui a été utilisé par défaut lorsque nous avons fait notre expérimentation. C'est pour ces deux raisons que nous avons choisi de l'étudier. [5], [28]

Chiffrement par flot ou par blocs ? `ChaCha20` est un algorithme de chiffrement par flot. Le chiffrement du message se fait bit par bit, avec une variation de la séquence chiffrante au fur et à mesure que les bits sont modifiés. Cette catégorie d'algorithme fait opposition aux algorithmes de chiffrement par blocs, qui, comme son nom l'indique, chiffre les données par blocs de plusieurs bits, et ce toujours de la même manière. De ce fait, les algorithmes de chiffrement par flots sont plus rapides. [29]

3.2.1 Opérations élémentaires et initialisation

Les opérations utilisées ici s'appliquent sur des nombres en base 2 de 32 bits, **que l'on dénommera « mots »** pour simplifier les notations.

Addition modulo 2^{32} L'addition se fait bit à bit, des bits de poids faibles vers les bits de poids forts. Des retenues peuvent se placer sur une colonne d'un bit de poids plus fort. Le modulo 2^{32} signifie que si le résultat fait plus de 32 bits, on le tronque, et on conserve un résultat sur 32 bits. En effet, on veut garder un nombre compris entre 0 et $2^{32} - 1$.
Exemple à petite échelle avec une addition de deux nombres en base 2 de 4 bits, modulo 2^4 :

$$\begin{array}{rr} & 1\,1\,1 \\ & 0\,1\,1\,1 \\ + & 1\,0\,1\,0 \\ \hline & 1\,0\,0\,0\,1 \\ \hline & 0\,0\,0\,1 \end{array}$$

OU exclusif Cette opération est plus simplement appelée XOR, de symbole $\bigoplus$ On va là aussi effectuer une opération bit à bits. En prenant le n-ième bit de chaque mot, appelons les A_n et B_n, on leur applique le OU exclusif dont la table de vérité est la suivante :

A_n	B_n	Bit chiffré
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 3.1 – Table de vérité du XOR

Rotation de n bits vers la gauche Cette opération, notée $mot <<< n$, permet de décaler chaque bit de n bits, des bits de poids faibles vers les bits de poids forts vers les bits de poids forts. Les bits dépassant les bits de poids forts reviennent vers les bits de poids faibles. Exemple à petite échelle, avec un mot de 4 bits : $1011 <<< 3 = 1101$.

Le petit boutisme Le boutisme désigne l'ordre dans lequel sont placés en mémoire les octets d'un nombre entier codé en binaire. Le petit boutisme s'oriente vers le placement des octets de poids faibles en premier. Pour l'entier 1110506459, représenté sur 4 octets en notation hexadécimale par '42 30 FB DB', on pourra lire, dans l'ordre des cases mémoires, 'DB FB 30 42'. [30]

Quart de tour

Le quart de tour est l'opération élémentaire de `ChaCha20`. Soit 4 mots a, b, c, d. On peut décomposer le quart de tour en quatre phases très similaires. Ces dernières utilisent les opérations élémentaires décrites précédemment. Par simplicité de lecture, nous allons regrouper ces quatre phases dans le tableau suivant :

1	2	3	4
$a = a + b$	$c = c + d$	$a = a + b$	$c = c + d$
$d = a \bigoplus d$	$b = b \bigoplus c$	$d = a \bigoplus d$	$b = b \bigoplus c$
$d = d <<< 16$	$b = b <<< 12$	$d = d <<< 8$	$b = b <<< 7$

TABLE 3.2 – Le quart de tour, opération élémentaire de ChaCha20 [28]

État interne

ChaCha20 possède ce que l'on appelle un état interne. Celui-ci peut être visualisé sous forme de matrice 4x4, où chaque élément représente un mot. On va pouvoir effectuer une opération quart de tour sur 4 de ces 16 mots, que l'on notera `QUART_TOUR(x, y, z, t)`, où x, y, z, t représentent l'indice du mot. A la fin du quart de tour, seule les mots d'indice x, y, z, t sont modifiés, les autres restent inchangés. [28]
La matrice suivante représente les indices de mot :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

Initialisation de l'état interne

`ChaCha20` a besoin de 4 éléments pour initialiser son état interne [28] :
- Une constante de 128 bits que l'on découpera en mots C1, C2, C3 et C4. Cette constante est toujours identique et vaut, en hexadécimal 0x61707865, 0x3320646e, 0x79622d32, 0x6b206574.
- Une clé de 256 bits, qu'on lira selon l'orientation du petit-boutisme (cf. §3.2.1) et que l'on découpera en mots K1, K2, K3, K4, K5, K6, K7, K8
- 32 bits servant de compteur, initialisés à 0. Ce sera un mot nommé CT.
- Un nonce (Number Used Once), c'est à dire un entier unique et temporaire, de 96 bits que l'on découpera en mots N1, N2, N3

Ces éléments vont permettre d'initialiser l'état interne de `ChaCha20`. Ainsi, on aura la matrice :

$$\mathrm{E}_d = \begin{pmatrix} C1 & C2 & C3 & C4 \\ K1 & K2 & K3 & K4 \\ K5 & K6 & K7 & K8 \\ CT & N1 & N2 & N3 \end{pmatrix}$$

3.2.2 Chiffrement et déchiffrement

Le but est d'obtenir une suite chiffrée de même longueur que les données à chiffrer. On va procéder par bloc de 64 octets.

Création d'une séquence chiffrante

On va appliquer sur l'état interne 8 fois la fonction `QUART_TOUR(x, y, z, t)`, avec à chaque fois des indices différents. Visuellement, par rapport à notre matrice, les indices (cf. §3.2.1) de chaque colonne et de chaque diagonale (gauche-haut → droite-bas) vont être passés en paramètre dans la fonction `QUART_TOUR(x, y, z, t)`. En clair, les appels effectués seront :

1. `QUART_TOUR(1,5,9,13)`
2. `QUART_TOUR(2,6,10,14)`
3. `QUART_TOUR(3,7,11,15)`
4. `QUART_TOUR(4,8,12,16)`
5. `QUART_TOUR(1,6,11,16)`
6. `QUART_TOUR(2,7,12,13)`
7. `QUART_TOUR(3,8,9,14)`
8. `QUART_TOUR(4,5,10,15)`

On répète cette opération 10 fois, et on obtient la matrice E_f avec des valeurs totalement différentes de celles que l'on avait à la base. On additionne ensuite les matrices pour obtenir $M = E_d + E_f$, là encore modulo 2^{32} (on rappelle qu'additionner deux matrices revient à additionner les nombres de même indice). On peut enfin concaténer les mots de la matrice, dans l'ordre de leurs indices. Chaque mot est rangé selon le petit-boutisme. On obtient alors une séquence chiffrante, qui peut être aussi bien obtenue par la machine qui chiffre que par la machine qui déchiffre [28]

Chiffrement des données

Une fois la séquence chiffrante obtenue, on va, par souci d'économie de mémoire, directement chiffrer les données à l'aide de cette séquence. Cette dernière étant indépendante des données, il n'y aurait pas de sens à l'envoyer. A partir de cette séquence, on va simplement chiffrer les données en effectuant un XOR bit à bit.

On incrémente le compteur et on répète cette opération n fois, où $n = \lfloor \frac{T}{64} \rfloor$ avec T le nombre de bits du message à chiffrer. Si le nombre de bits du message à chiffrer n'est pas un multiple de 64, alors il reste encore quelques bits (moins de 64) non chiffrés. On incrémente encore le compteur et on recrée un bloc de chiffrement, qui nous donne une séquence chiffrante. On effectue enfin le XOR bit à bit avec les bits restants. Ainsi, toutes les données sont chiffrées.

Déchiffrement

Le déchiffrement se déroule de la même manière que pour le chiffrement. On calcule la séquence chiffrante par blocs de 64 octets. Seule différence, au lieu de chiffrer le message en clair avec le XOR, on va déchiffrer le message chiffré avec le XOR. En effet $(B \bigoplus S) \bigoplus S = B$. [28]
On peut montrer cela avec une table de vérité :

Bit en clair	Bit chiffrant	Bit chiffré	Bit chiffrant	Bit déchiffré
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	0	1	1

Table 3.3 – Table de vérité du XOR

4 | Bilan Projet

Tout d'abord, le choix de ce sujet n'a pas été une évidence, nous avons choisi d'orienter notre TIPE sur la cryptographie, mais sans réelle idée du domaine d'application. Pour nous aider, nous nous sommes documentés sur les grands principes de la cryptographie, ceci a débouché sur l'étude de différents domaines. Comme les terminaux de paiement ou le cryptage et l'authentification de mails ou encore les signatures électroniques. C'est l'utilisation de GitHub qui nous a orienté vers ce choix de sujet en désirant utiliser le protocole SSH pour connecter notre ordinateur à GitHub.

Pour réaliser notre projet, nous avons mis en place un environnement numérique pour le travail collaboratif. Tout d'abord, par le déploiement d'une équipe Teams qui nous a permise de centraliser nos visioconférences, nos communications ainsi que nos outils de travail. Ensuite, nous avons utilisé OneNote pour centraliser nos recherches ainsi que les notes lors de nos réunions et points avec notre tuteur. Miro a été également un outil important pour créer des cartes mentales pour découvrir les sujets lors de nos recherches. Pour planifier les différentes étapes du projet et suivre sa progression. Enfin, nous avons utilisé l'outil de la suite Office intégré à notre Teams, planner.

Ce rapport a été rédigé à l'aide de LaTeX, nous souhaitions découvrir cet outil pour la rédaction d'articles scientifiques de qualité. Nous avons pour cela créé notre projet sur Overleaf pour pouvoir collaborer. Nous avons pu découvrir de nombreuses fonctionnalités offertes par LaTeX avec des nombreux packages à notre disposition. La gestion des sources de nos recherches a été réalisée avec Zotero pour obtenir un meilleur suivi et faciliter la réalisation de notre bibliographie avec l'export d'un fichier `.bib`. Ces compétences acquises nous seront utiles pour réaliser nos futurs documents scientifiques.

Ce projet a été l'occasion d'utiliser et d'approfondir des notions vues au sein du CITISE. Ceci a notamment été le cas avec l'étude de trames réseaux qui nous a permis d'approfondir certaines notions vues en cours de réseaux. L'étude de ce protocole nous permet également de mieux comprendre l'utilisation que nous en faisons en AT33 (R32 InfoSpé) pour la communication entre notre Raspberry et notre ordinateur. Pour conclure, ce projet nous a permis de découvrir le domaine de la cryptographie, de renforcer nos connaissances, tant au niveau scientifique que pour la gestion de projet, sans oublier l'étude de la sécurité informatique avec une approche technique.

5 | Conclusion

Au cours de ce rapport, nous avons pu voir les deux types de cryptographie (asymétrique puis symétrique) de manière concrète, implémentées dans le protocole SSH, mais aussi tout le déroulement de ce dernier.

En commençant par l'analyse des trames, nous avons observé les échanges le réseau au travers des différentes couches du modèle OSI. Durant la phase de hand-shake, chaque partie est arrivée aux mêmes choix d'algorithmes, puis à la même clé cryptographique, sans échanger « en clair » ces informations.

Il a été montré que l'algorithme de Diffie-Hellman permet d'échanger cette clé de manière sécurisée, au travers de paramètres privés et publics. Les opérations mathématiques et l'ordre de grandeur des nombres impliqués dans cet échange sont tels qu'il est impossible pour un attaquant de deviner le secret échangé en un temps raisonnable : c'est le problème du logarithme discret. La génération des paramètres publics fait intervenir la notion de groupe cyclique.

Les algorithmes cryptographiques asymétriques permettent aussi bien de chiffrer des données que d'authentifier une machine, bien que l'authenticité des clés publiques ne puisse pas toujours être garantie. Le chiffrement RSA par l'arithmétique modulaire et le problème de factorisation de grands nombres premiers, permettent par des clés publiques et privées d'authentifier le serveur et le client.

La sécurité avec les algorithmes cryptographiques symétriques lors de la transmission de la clé est garanti par l'échange de Diffie-Hellman. Avec le chiffrement symétrique, il est possible au travers d'opérations simples de chiffrer des données de manière très efficaces. L'algorithme `ChaCha20` offre en particulier une belle efficacité au niveau mémoire.

Avec les recherches menées sur les ordinateurs quantiques, des prouesses pourraient être réalisées. Ce type d'architecture offre une puissance de calcul nettement supérieure à celle des ordinateurs à architecture classique, et seraient seraient à même de réduire à néant la sécurité des algorithmes cryptographiques. Nous pouvons alors nous interroger sur le devenir de la sécurité informatique, lorsque les ordinateurs quantiques seront opérationnels pour de telles applications.

Bibliographie

[1] J.-M. Jancovici, *Le Monde sans fin*, Dargaud. oct. 2021.

[2] *Attaques des systèmes*, fr. adresse : https://www.techniques-ingenieur.fr/base-documentaire/technologies-de-l-information-th9/cybersecurite-attaques-et-mesures-de-protection-des-si-42313210/attaques-des-systemes-h5832/ (visité le 17/12/2023).

[3] HAJJEH Ibrahim et BADRA Mohamad, "Le protocole SSH," fre, *Techniques de l'ingénieur Sécurité des systèmes d'information*, t. base documentaire : TIP440WEB, n° ref. article : h5235, oct. 2006, Publisher : Editions T.I. doi : 10.51257/a-v1-h5235. adresse : https://www.techniques-ingenieur.fr/base-documentaire/technologies-de-l-information-th9/cryptographie-authentification-protocoles-de-securite-vpn-42314210/le-protocole-ssh-h5235/.

[4] C. M. Lonvick et T. Ylonen, "The Secure Shell (SSH) Protocol Architecture," Internet Engineering Task Force, Request for Comments RFC 4251, jan. 2006, Num Pages : 30. doi : 10.17487/RFC4251. adresse : https://datatracker.ietf.org/doc/rfc4251 (visité le 17/12/2023).

[5] T. Cadegros et S. Prost, *Etude des trames SSH avec Wireshark*, oct. 2023.

[6] F. Gérossier, *Cours de réseaux - DUT GEII CITISE - S3*, 2020.

[7] C. M. Lonvick et T. Ylonen, "The Secure Shell (SSH) Transport Layer Protocol," Internet Engineering Task Force, Request for Comments RFC 4253, jan. 2006, Num Pages : 32. doi : 10.17487/RFC4253. adresse : https://datatracker.ietf.org/doc/rfc4253 (visité le 03/11/2023).

[8] P.-A. Fouque, "Cryptographie appliquée," *Sécurité des systèmes d'information*, nov. 2003. doi : 10.51257/a-v2-h5210. adresse : https://www.techniques-ingenieur.fr/doi/10.51257/a/v2/h5210 (visité le 22/09/2023).

[9] *Cryptographie asymétrique*, fr. adresse : https://www.techniques-ingenieur.fr/base-documentaire/technologies-de-l-information-th9/cryptographie-authentification-protocoles-de-securite-vpn-42314210/cryptographie-appliquee-h5210/cryptographie-asymetrique-h5210niv10003.html (visité le 18/12/2023).

[10] G. CASTAGNOS, *Cours de cryptologie avancée*, sept. 2023. adresse : https://www.math.u-bordeaux.fr/~gcastagn/CryptoAv/CoursCryptoAv-23-24.pdf (visité le 02/11/2023).

[11] R. Serrano, C. Duran, M. Sarmiento, C.-K. Pham et T.-T. Hoang, "ChaCha20–Poly1305 Authenticated Encryption with Additional Data for Transport Layer Security 1.3," *Cryptography*, t. 6, p. 30, juin 2022. doi : 10.3390/cryptography6020030.

[12] *Groupe*. adresse : https://www.bibmath.net/dico/index.php?action=affiche&quoi=./g/groupe.html (visité le 16/11/2023).

[13] *Sous-groupe*. adresse : https://www.bibmath.net/dico/index.php?action=affiche&quoi=./s/sousgroupe.html (visité le 16/11/2023).

[14] C. Polytechnique, *Groupe.pdf*. adresse : http://www.cmls.polytechnique.fr/perso/haringck/DocEv/cours/Groupe (visité le 18/11/2023).

[15] E. Ellipses, *Groupes monogènes. Groupes cycliques. Exemples*. adresse : https://www.editions-ellipses.fr/PDF/9782340017870_extrait.pdf (visité le 21/11/2023).

[16] A. L. Gluher, "PROBLÈME DU LOGARITHME DISCRET APPLIQUÉ À LA CRYPTANALYSE SUR COURBES ELLIPTIQUES : ALGORITHME MOV,"

[17] Centre Henri Lebesgue, *David Lubicz - Le problème du logarithme discret*, sept. 2017. adresse : https://www.youtube.com/watch?v=Ds-kHs6yb5E (visité le 02/11/2023).

[18] ANSSI, *GUIDE DES MÉCANISMES CRYPTOGRAPHIQUES*, jan. 2020.

[19] N. Provos, M. Friedl et W. Simpson, *OpenSSH Manual - Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol*, English. adresse : https://www.ietf.org/rfc/rfc4419.txt (visité le 17/12/2023).

[20] M. Friedl, N. Provos et W. Simpson, "Échange de groupe Diffie-Hellman pour le protocole de couche Transport Secure Shell (SSH)," adresse : https://www.academia.edu/2891047/%C3%89change_de_groupe_Diffie_Hellman_pour_le_protocole_de_couche_Transport_Secure_Shell_SSH_.

[21] R. Perlman, "An overview of PKI trust models," *IEEE Network*, t. 13, n° 6, p. 38-43, nov. 1999, Conference Name : IEEE Network, issn : 1558-156X. doi : 10.1109/65.806987. adresse : https://ieeexplore.ieee.org/abstract/document/806987 (visité le 17/12/2023).

[22] T. Cadegros et S. Prost, *Authentification du serveur*, déc. 2023.

[23] T. Cadegros et S. Prost, *Authentification du client par SSH-PublicKey*, fr, déc. 2023.

[24] S. Rostam, *Groupe multiplicatif d'un corps fini*, fr. adresse : https://perso.univ-rennes1.fr/ludovic.marquis/enseign/2018-19/THGG_2018/cor_exo4_F1.5.pdf (visité le 12/05/2023).

[25] T. Cormen, *Algorithmes : Notions de base - Chap.8 p.151*. Dunod, 2013, isbn : 978-2-10-070151-3. adresse : http://unr-ra.scholarvox.com/book/88817457 (visité le 12/12/2023).

[26] D. L. Hostalot, "Attaque par factorisation contre RSA," fr, adresse : https://www.academia.edu/53891839/Attaque_par_factorisation_contre_RSA.

[27] C. Bouvier, "Algorithmes pour la factorisation d'entiers et le calcul de logarithme discret," fr, thèse de doct., Université de Lorraine, juin 2015. adresse : https://theses.hal.science/tel-01751648 (visité le 13/12/2023).

[28] Y. Nir et A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," Internet Engineering Task Force, Request for Comments RFC 7539, mai 2015, Num Pages : 45. doi : 10.17487/RFC7539. adresse : https://datatracker.ietf.org/doc/rfc7539 (visité le 12/12/2023).

[29] *Cryptographie symétrique*, fr. adresse : https://www.techniques-ingenieur.fr/base-documentaire/technologies-de-l-information-th9/cryptographie-authentification-protocoles-de-securite-vpn-42314210/cryptographie-appliquee-h5210/cryptographie-symetrique-h5210niv10002.html (visité le 18/12/2023).

[30] *Définition de boutisme | Dictionnaire français*, fr. adresse : https://www.lalanguefrancaise.com/dictionnaire/definition/boutisme (visité le 17/12/2023).

Table des figures

Télécom Saint-Etienne
École affiliée institut mines télécom
École interne à l'université Jean-Monnet

20 décembre 2023

Cryptographie du protocole SSH

Projet TIPE 2023 - CITISE 2

Thomas Cadegros, thomas.cadegros@telecom-st-etienne.fr
Sylvain Prost, sylvain.prost@telecom-st-etienne.fr

Résumé

Le protocole Secure Shell (SSH) est utilisé pour la communication entre un serveur et un client. Nous le retrouvons dans plusieurs services comme GitHub, il permet de chiffrer les échanges entre le client et le serveur ainsi que de les authentifier. Plusieurs méthodes cryptographiques interviennent pour mettre en place le chiffrage. A partir de l'étude des trames de la connexion d'un client à un serveur, ce document décrypte les différentes étapes amenant à des échanges cryptés et décrit leurs fonctionnements. Il y est notamment décrit l'échange de clés permettant la mise en place de la cryptographie symétrique, avec la description du fonctionnement de l'échange de clés de Diffie-Hellman. L'authentification du client et du serveur à l'aide de la cryptographie asymétrique, avec l'étude du chiffrement RSA, y est aussi décrite. Finalement, il y est décrit le chiffrement des données par la cryptographie symétrique ainsi que le fonctionnement du chiffrement ChaCha20.

Mots-Clés : SSH, Secure Shell, Cryptographie, Cryptographie Symétrique, Cryptographie Asymétrique, Echange de Diffie et Hellman, RSA, ChaCha20

Abstract

The Secure Shell (SSH) protocol is used to communicate between a server and a client. For example, GitHub uses SSH to communicate with your computer. The SSH allows to encode frames and authenticate the server and the client. With the frames of the connection of a client to a server, this document decrypts the different steps to implement the symmetric cryptography. Firstly, the operation of the key exchange to achieve this goal and the operation of the Diffie-Hellman key exchange. Secondly, the authentication of the server and the client is discribed with the operation of the RSA encryption. Finally, the operation of symetric encryption of the frames and ChaCha20 encryption is studied.

Keywords : SSH, Secure Shell, Cryptography, Symmetric Cryptography, Asymmetric Cryptography, Diffie-Hellman exchange, RSA, ChaCha20