Thomas DAGIER Quentin ROD

Université d’été : Rapport de Travaux Pratiques

#### Cours de sécurité encadré par Noria Foukia 24 Aout 2020 – 11 Septembre 2020

Table des matières

[Premier Travail Pratique : Chiffrement IDEA 4](#_bookmark0)

[Introduction 4](#_bookmark1)

[Objectif du TP 4](#_bookmark2)

[Implémentation de l’algorithme: 5](#_bookmark4)

[Les opérations à réaliser dans les rounds 5](#_bookmark5)

[Type et format des données 5](#_bookmark6)

[Lecture du fichier texte 5](#_bookmark7)

[La création des clefs de chiffrement 6](#_bookmark9)

[Chiffrement IDEA Simplifié: 6](#_bookmark10)

[La création des clefs de déchiffrement 6](#_bookmark11)

[Inverse modulaire par rapport à l’addition 8](#_bookmark13)

[Inverse modulaire par rapport à la multiplication 8](#_bookmark14)

[Déchiffrement IDEA Simplifié: 8](#_bookmark15)

[Enregistrement du déchiffrement dans le fichier 9](#_bookmark16)

[Démonstration du programme 9](#_bookmark18)

[Conclusion 10](#_bookmark22)

[Second Travail Pratique : Certification SSL 11](#_bookmark23)

[Introduction 11](#_bookmark24)

[La création des certificats 11](#_bookmark27)

[Création du certificat serveur 11](#_bookmark28)

[Création d’un fichier de demande de signature de certificat CSR 13](#_bookmark33)

[Création du certificat de l’autorité de certification 13](#_bookmark35)

[Signature des certificats 14](#_bookmark37)

[Signature du certificat serveur par l’autorité de certification 14](#_bookmark38)

[Installation du certificat d’autorité de certification 15](#_bookmark40)

[Installation et configuration d’Apache2 15](#_bookmark42)

[Phase de test 17](#_bookmark47)

[Conclusion 17](#_bookmark49)

[Troisième Travail Pratique : Pretty Good Privacy 18](#_bookmark50)

[Schéma de fonctionnement 18](#_bookmark51)

[Installation de GnuPG 19](#_bookmark53)

[Installation des clés GPG 19](#_bookmark56)

[Exportation de la clé publique et envoie de la clé aux correspondants 22](#_bookmark62)

[Importation de la clef publique depuis ses correspondants pour stockage interne 23](#_bookmark65)

[Envoie et réception de message crypté 24](#_bookmark67)

[Création de la signature 25](#_bookmark71)

[Vérification de la signature 25](#_bookmark72)

[Questions complémentaires 26](#_bookmark74)

[Conclusion 26](#_bookmark75)

[Quatrième Travail Pratique : Firewall & Services 27](#_bookmark76)

[Firewall sous CentOS 7 27](#_bookmark77)

[Installation de firewalld 27](#_bookmark78)

[Observation des zones 27](#_bookmark82)

[Observation des fichiers de services 28](#_bookmark86)

[Création d’une nouvelle zone 30](#_bookmark90)

[Configuration de la zone SSIzone 30](#_bookmark94)

[Activation de la zone 31](#_bookmark97)

[Test de la zone 32](#_bookmark101)

[VSFTPD sous CentOS 7 32](#_bookmark103)

[Installation de VSFTPD (Very Secure File Transfer Protocol Daemon) 32](#_bookmark104)

[Ouverture des ports pour l’utilisation de FTP 32](#_bookmark106)

[Configuration du serveur FTP 33](#_bookmark108)

[Préparation de l’accès au service FTP (À faire par 2 obligatoirement) 34](#_bookmark110)

[Test de l’accès au service FTP du binôme 34](#_bookmark112)

[Sécurisation supplémentaire 35](#_bookmark116)

[Conclusion 37](#_bookmark120)

[Table des illustrations 38](#_bookmark121)

[Sources 39](#_bookmark122)

# Premier Travail Pratique : Chiffrement IDEA

## Introduction

IDEA signifie “International Data Encryption Algorithm”. C’est un algorithme de [chiffrement](https://fr.wikipedia.org/wiki/Chiffrement_sym%C3%A9trique) symétrique conçu par [Xuejia Lai](https://fr.wikipedia.org/wiki/Xuejia_Lai) et [James Massey](https://fr.wikipedia.org/wiki/James_Massey) en 1991.

Le principe de l’algorithme est d’utiliser une clef d’un nombre de 𝑋 bits pour chiffrer un message de taille bits. Cette même clef sera utilisée pour déchiffrer le message.

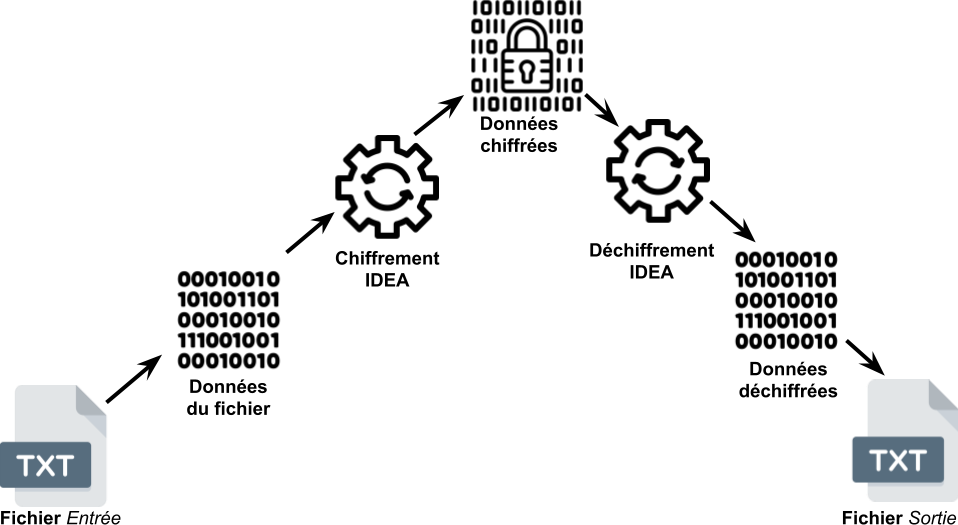
Le chiffrement et le déchiffrement s’effectuent en plusieurs “rounds”. Un round consiste en une suite de 14 opérations à effectuer. Les étapes sont identiques pour le chiffrement et le déchiffrement. À chaque round, le message et la clef sont respectivement découpés en 4 et en 8 parties.

On utilise généralement une clef de 128 bits et donc un message sur 64 bits. Il y a 4 rounds

½ pour le chiffrement et tout autant pour le déchiffrement. La “½” signifie que l’on effectue uniquement les 4 premières opérations du dernier round. Dans le cadre de ce travail pratique nous réalisons l’algorithme IDEA Simplifié soit une clef de 32 bits, un message de 16 bits et 4 rounds ½ . Les opérations restent identiques.

## Objectif du TP

L’objectif de ce travail pratique consiste dans un premier temps à chiffrer avec l’algorithme IDEA Simplifié les données contenues dans un fichier texte. Ensuite, nous devons déchiffrer ces données et les enregistrer dans un fichier texte : le contenu de départ devant être le même que celui inscrit dans le fichier texte de sortie.



*Figure 1 Fonctionnement global du programme*

## Implémentation de l’algorithme:

### Les opérations à réaliser dans les rounds

- Addition modulaire:

Soit 𝐴 et 𝐵 deux nombres entiers relatifs et 𝑅 le résultat de l’addition 𝑚𝑜𝑑 𝑛 de 𝐴 et 𝐵 :

𝑅 = (𝐴 + 𝐵) 𝑚𝑜𝑑 𝑛 = (𝐴 𝑚𝑜𝑑 𝑛 + 𝐵 𝑚𝑜𝑑 𝑛) 𝑚𝑜𝑑 𝑛

Le résultat 𝑅 appartient à 𝑍

𝑁𝑍

= {0, … , 𝑛 − 1}

Dans le cas de l’algorithme IDEA Simplifié, 𝑛 = 24 = 16

- Multiplication modulaire :

Soit 𝐴 et 𝐵 deux nombres entiers relatifs et 𝑅 le résultat de la multiplication 𝑚𝑜𝑑 𝑛 de 𝐴 et 𝐵 :

𝑅 = (𝐴 ∗ 𝐵) 𝑚𝑜𝑑 𝑛 = (𝐴 𝑚𝑜𝑑 𝑛 ∗ 𝐵 𝑚𝑜𝑑 𝑛) 𝑚𝑜𝑑 𝑛

Le résultat 𝑅 appartient à 𝑍

𝑁𝑍

= {0, … , 𝑛 − 1}

Dans le cas de l’algorithme IDEA Simplifié, 𝑛 = 24 + 1 = 17

Note : à chaque fois que le résultat est 0, cela correspond à la valeur décimale 16. Cela permet de représenter sur 4 bits les valeurs de 1 à 16.

### Type et format des données

Nous avons réfléchi pour savoir comment stocker sous forme informatique le message, la clef ainsi que les différents sous-éléments. Nous savons que le message et la clef sont en binaire. Pour stocker ces données en Python, nous avons choisi de les représenter sous forme de chaîne de caractères. Par exemple, *0b1010110* sera stocké “*1010110*”. Cela nous permet d’effectuer aisément des décalages ainsi que l’opération logique XOR.

Nous avons alors créé des fonctions nous permettant de jongler entre le format binaire, en chaîne de caractère et la valeur décimal.

Par exemple, **convert\_string\_binary\_to\_int(numberString)**

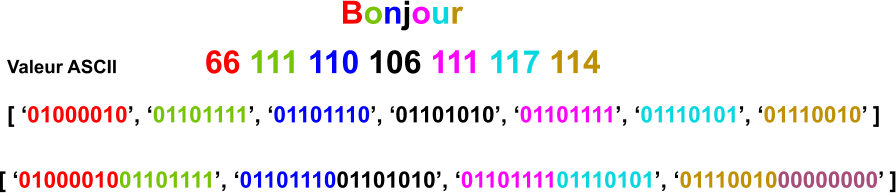
ou **convert\_int\_to\_string\_binary(numberInt)** nous permettent de passer d’une base à l’autre pour faire les calculs et stocker les données.

### Lecture du fichier texte

Chaque caractère est lu un à un et sa valeur en Ascii est récupérée. Ce nombre est alors converti au format binaire sous forme d’une chaîne de caractère. Un ‘0’ est ajouté comme MSB afin d’obtenir un nombre de 8 bits et donc 8 caractères. Ces chaînes sont stockées à la suite dans une liste.

L’algorithme IDEA Simplifié nécessite un message de 16 bits. Par conséquent, chaque chaîne de caractères du tableau est concaténée deux à deux afin de faire une chaine sur 16 bits. Toutes les nouvelles chaînes sont stockées à la suite dans un nouveau tableau.

Si le nombre de caractères est impair, le dernier est concaténé avec la valeur “00000000“. Cela représente en ASCII le caractère NULL.



*Figure 2 Opérations lors de la lecture d'un fichier texte*

Le chiffrement sera alors effectué sur chacune des valeurs du dernier tableau.

## La création des clefs de chiffrement

Pour le chiffrement comme le déchiffrement, 6 sous-clés de 4 bits sont utilisées par “round”. L’ensemble de ces dernières est trouvable à partir de la clef initiale sur 32 bits. La fonction **fill\_array(keyString)** réalise ceci et retourne un tableau en deux dimensions. Les lignes représentent les rounds et les colonnes le numéro de la clef. On a donc un tableau de 6x5 cases en sachant que les deux dernières cases de la dernière ligne ne sont pas utilisées comme le dernier round ne se fait qu’à moitié.

La clef étant sur 32 bits, cela fait 8 sous-clés. Comme uniquement 6 sont utilisées pour un round, les deux dernières sont utilisées pour le round suivant. Par exemple, au premier round, la 7ème et la 8ème correspondent respectivement à la première et deuxième sous-clef du deuxième round. Pour générer la clef du prochain round, un décalage de 6 bits vers la gauche est effectué sur la clef de 32 bits. On continue comme ceci jusqu’à avoir généré l’intégralité des clefs nécessaires.

## Chiffrement IDEA Simplifié:

La fonction utilisée pour chiffrer un nombre binaire de 16 bits sous forme de chaîne caractères avec une clef sur 32 bits est **encryption\_idea(key32bits, text16bits).** Cette dernière va appliquer les opérations des 4 rounds ½ et retourner le message chiffré soit une chaine de caractère contenant 16 bits. Pour réaliser cela, la fonction génère le tableau des clefs grâce à la fonction **fill\_array(keyString)**. Les étapes sont les mêmes à chaque round donc on peut simplement faire une boucle et réaliser ces étapes pour les 4 rounds.

L’ensemble des valeurs du tableau issu de la lecture du fichier sont parcourues une à une et chiffrées avec **encryption\_idea(key32bits, text16bits)**. Chaque valeur retournée est stockée à la suite dans un tableau contenant les nombres chiffrés sous forme de chaînes. Ce dernier tableau sera ensuite parcouru de la même manière pour déchiffrer le message.

### La création des clefs de déchiffrement

Les clefs de déchiffrement se basent sur les clefs de chiffrements. Il s’agit soit d’une même clef soit de l’inverse par rapport à l’addition ou à la multiplication.

0 3 0 4 0

44 −1

Par exemple respectivement : 𝐾4 = 𝑍4 ; 𝐾2 = -𝑍2 ; 𝐾3 = (𝑍3)

0

On pose donc 𝐾4 la clef de déchiffrement à utiliser à l’étape 4 du round 0 et 𝑍 la clef de

chiffrement à l’étape 4 du round 3 avec “-” signifiant inverse par rapport à l’addition et “ −1 ” inverse par rapport à la multiplication. Cela dépend de l’opération mathématique utilisée avec la clef pendant le round. Par exemple, la clef 𝐾 d’indice 3 est utilisée au round 4 pour effectuer une multiplication donc “ −1 ”.

Également, la clef 𝐾 d’indice 2 est utilisée au round 3 pour faire une addition donc “-”. En ce qui concerne les clefs d’indice 4 ou 5, aucun inverse n’est à calculer.

Comme nous pouvons le voir, il suffit d’effectuer une sorte de symétrie pour trouver la correspondance entre les clefs. Il n’est donc pas très compliqué de trouver un lien entre le tableau des clefs , 𝐾 (déchiffrement) et 𝑍 (chiffrement).

𝑖

On rappelle donc que 𝐾𝑗 est la clef de déchiffrement sur laquelle on aura déjà appliqué une

𝑖

modification si besoin. On pose donc 𝐻𝑗 , la clef correspondante de déchiffrement de l’étape i et du round j qui n’a pas encore été modifiée.

Cela signifie qu’uniquement la symétrie est effectuée et qu’aucun inverse n’a été calculé.

𝑖

Pour connaitre 𝐾𝑗 il faut encore effectuer un inverse modulaire ou non suivant le cas rencontré. On obtient alors :

Si 𝑗 > 4 :

𝑖

𝐻𝑗 = 𝑍

Autrement :

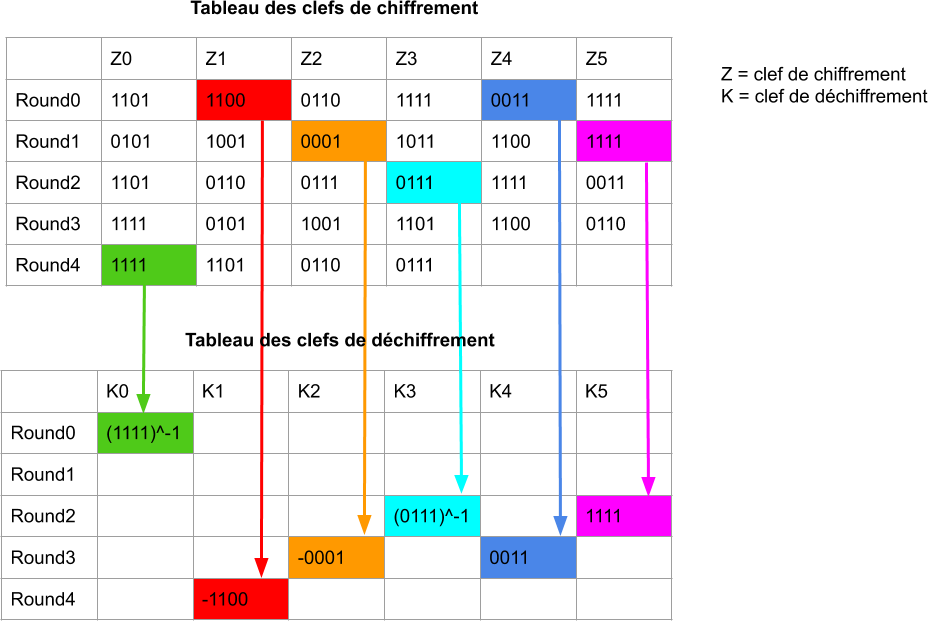
𝑖

𝐻𝑗 = 𝑍

4−𝑖

𝑗

4−𝑖−1

𝑗

*Figure 3 Correspondance entre les clefs de chiffrements et de déchiffrements*

Afin de connaître la clef de chiffrement correspondante, on utilise la fonction

##### get\_correspondant\_key\_from\_encryption (keyArray, i, j).

Celle-ci demande respectivement le tableau de clef de chiffrement ainsi que l’indice du round et celui de la clef de déchiffrement. Nous allons utiliser la valeur de retour de cette fonction pour calculer si besoin l’inverse et ensuite être utilisée lors de l’étape de déchiffrement. Cette fonction effectue donc uniquement la symétrie et ne calcule pas l’inverse.

### Inverse modulaire par rapport à l’addition

Soit 𝐴 et 𝐵 deux nombres entiers relatifs et 𝑛 un entier naturel.

𝐵 est inverse de 𝐴, 𝑚𝑜𝑑 𝑛 si et seulement si (𝐴 + 𝐵) 𝑚𝑜𝑑 𝑛 = 0

On peut observer qu’il est assez aisé de trouver l’inverse modulaire par rapport à l’addition. Si nous connaissons 𝐴 et 𝑛, il suffit de faire 𝐵 = (𝑛 − 𝐴) 𝑚𝑜𝑑 𝑛

Dans le cas de l’algorithme IDEA simplifié, 𝑛 = 16.

### Inverse modulaire par rapport à la multiplication

Soit 𝐴 et 𝐵 deux nombres entiers relatifs et 𝑛 un entier naturel.

𝐵 est inverse de 𝐴, 𝑚𝑜𝑑 𝑛 si et seulement si (𝐴 ∗ 𝐵) 𝑚𝑜𝑑 𝑛 = 1

Dans le cas de l’algorithme IDEA simplifié, 𝑛 = 17.

Cependant, calculer l’inverse modulaire par rapport à la multiplication est plus complexe. Pour cela, nous avons mis en place l’algorithme d’Euclide étendu qui grâce aux coefficients de Bézout nous permet de trouver cet inverse modulaire.

En effet, si 𝐴 a un inverse 𝑚𝑜𝑑 𝑛 alors 𝑝𝑔𝑐𝑑(𝐴, 𝑛) = 1. Par conséquent, il existe deux entiers relatifs 𝑈 et 𝑉 appelés coefficients de Bézout tel que 𝑈 ∗ 𝐴 + 𝑉 ∗ 𝑛 = 1. L’inverse de 𝐴 est donc 𝑈. Nous nous intéressons donc uniquement à calculer 𝑈.

L’algorithme d’Euclide est utilisé pour calculer le 𝑝𝑔𝑐𝑑 entre deux nombres. On effectue une division euclidienne du plus grand par le plus petit. Ensuite, le diviseur devient le dividende et le reste devient le diviseur.

Le dernier reste non nul est le 𝑝𝑔𝑐𝑑. À chaque étape notée 𝑖 nous calculons aussi le coefficient de Bézout 𝑝. Nous savons que pi = pi-2 - pi-1 qi-2 (𝑚𝑜𝑑 𝑛),

que p0 = 0 et p1 = 1 avec q correspondant au quotient.

Pour que 𝑝 soit égale à 𝑈, il faut que le dernier reste non nul de la division euclidienne valle 1. Cette étape est notée k. Alors, pk+2 = U.

Cependant, cela ne fonctionne pas pour les nombres qui n’ont pas d’inverse. Dans notre cas,

𝑛 vaut toujours 17 et est premier. Par conséquent l’ensemble des nombres de 1 à 16 sont inversibles. Cela fonctionne alors dans tous les cas.

## Déchiffrement IDEA Simplifié:

La fonction utilisée pour déchiffrer un nombre binaire de 16 bits sous forme de chaîne caractères avec une clef sur 32 bits est **decryption\_idea(key32bits, cypher16bits)**. Elle effectue les mêmes opérations que pour le chiffrement avec cependant des clefs qui diffèrent. Il est également créé le tableau des clefs de chiffrements avec **fill\_array(keyString)**.

L’ensemble des valeurs du tableau issu du chiffrement sont parcourues une à une et déchiffrées avec **decryption\_idea(key32bits, cypher16bits)**. Chaque valeur retournée est stockée à la suite dans un tableau contenant les nombres déchiffrés sous forme de chaînes.

À chaque fois que l’on doit connaître la valeur d’une clef de déchiffrement, la fonction **calculate\_decryption\_key(keyArray, i, j, operationType)** est utilisée. Elle renvoie la valeur de la clef de déchiffrement j du round i. La fonction a besoin du tableau des clefs de chiffrements ainsi que l’opérateur par lequel il faut calculer l’inverse (addition ou multiplication).

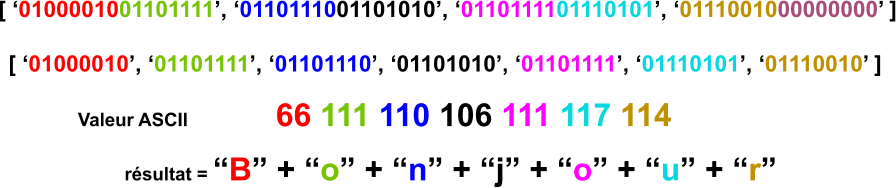
Cette dernière fonction récupère dans un premier temps la valeur de la clef correspondante avec **get\_correspondant\_key\_from\_encryption(keyArray, i, j)** et calcule ensuite, si besoin, son inverse. Pour cela, une simple soustraction est effectuée s’il s’agit d’une addition sinon, la fonction **extended\_euclidian\_algorithm(number, modulo)** est utilisée dans le cas de la multiplication. C’est alors la valeur calculée de l’inverse qui est retournée.

## Enregistrement du déchiffrement dans le fichier

À la fin de l’étape de déchiffrement, on obtient alors une liste contenant des chaînes de caractères sur 16 bits décodées. Ces chaînes binaires correspondent chacune à deux caractères que l’on va devoir stocker dans le fichier de sortie. À l’inverse de l’étape de lecture du fichier texte, nous allons ici de-concaténer chaque chaîne sur 16 bits en deux chaînes sur 8 bits. Chaque chaîne de 8 bits est stockée à la suite dans une liste uniquement si la valeur binaire correspondante est différente de 0. En effet, on sait qu’une chaîne correspondant à la valeur 0 n’est pas dans le message de départ et a donc été ajouté pour compléter une chaîne lors du chiffrement.

Une fois le tableau de chaînes de 8 bits récupéré, on utilise la fonction **convert\_string\_binary\_to\_int(numberString)** pour avoir la valeur ASCII de chacune des chaînes.

La dernière étape est donc de stocker dans une variable le message décodé en prenant soin d’utiliser la fonction **chr()** pour convertir une valeur ASCII en caractère.



*Figure 4 Operations lors de l'écriture dans un fichier texte*

Enfin, on écrit le contenu de cette variable dans le fichier de sortie au format texte.

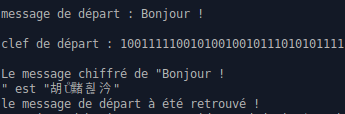
Il aurait aussi été possible de stocker dans un autre fichier le résultat correspondant au message encodé. Nous avons privilégié la mise en place des algorithmes. L’écriture du message encodé se serait faite exactement de la même manière à la différence que les caractères ne seraient pas compréhensibles.

## Démonstration du programme

Pour lancer le programme, on utilise la commande : “python3 <input\_file.txt> <output\_file.txt>”. Le premier argument correspond au fichier texte contenant le message initial et le second correspond au fichier qui contiendra le message décodé.

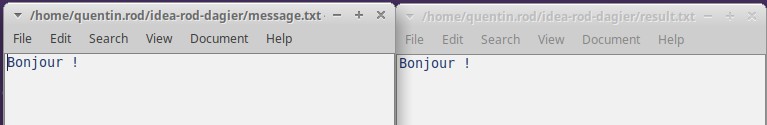
Pour réaliser le programme, nous avons écrit une librairie IDEA qui contient toutes les fonctions utilisées au cours de l’algorithme.

Nous avons créé un fichier nommé “message.txt” qui contient “Bonjour!\n”. Nous exécutons le programme en entrant dans le terminal : “python3 main.py message.txt result.txt”. Le programme va alors chiffrer puis déchiffrer le texte et l’écrire dans “result.txt”. Il est affiché sur le terminal le message de départ, la clef générée aléatoirement mais aussi si le message déchiffré est bien identique au message à chiffrer. Dans cet exemple-ci , nous avons affiché au format ASCII la valeur du texte chiffré.



*Figure 5 Exécution du code et affichage du terminal*

Afin d’être certain que les fichiers “result.txt” et “message.txt” soient bien identiques, nous comparons les deux fichiers visuellement.



*Figure 6 Comparaison fichier initial et résultat*

Nous pouvons observer qu’il n’y a bien aucune différence. Cependant, il se peut que certains caractères soient présents mais pas affichés. Afin d’en avoir la certitude nous effectuons la commande “diff” permettant de comparer deux fichiers.



*Figure 7 Vérification des différences entre les deux fichiers*

En effectuant la commande, aucune différence n’est trouvée, affirmant que les textes sont identiques.

## Conclusion

La réalisation de cet algorithme nous a permis de comprendre aussi bien de manière pratique que théorique d’une part un nouvel algorithme de chiffrement et d’autre part, les bases de l’algorithme DES qui est très utilisé aujourd’hui.

Concernant nos points forts et faiblesses sur la réalisation de ce programme, nous avons rapidement délimité ce dont nous avions besoin pour avancer rapidement sans avoir à revenir sur le code. Une étape clef a été de décrire le fonctionnement de chaque fonction dans le code et de bien séparer chaque tâche pour ne pas tout mélanger.

En revanche, nous avons passé plus de temps sur l’algorithme d’Euclide étendu car nous voulions un résultat efficace tout en comprenant correctement ce que nous faisions.

Second Travail Pratique : Certification SSL

## Introduction

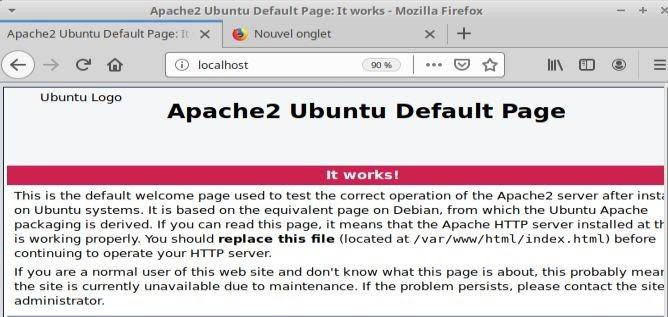
Afin de réaliser ce travail pratique nous avons mis en place une machine virtuelle Ubuntu sur les ordinateurs de l’école. Le nom d’hôte de la machine est “kentom-VirtualBox”.

La première étape consiste à installer Apache2. Pour cela, nous entrons dans le terminal en mode root avec la commande : “sudo su root”. Nous utilisons alors la commande “apt-get install apache2”.



*Figure 8 Installation Apache2*

Afin de vérifier le bon fonctionnement d’Apache2, nous nous rendons sur un navigateur web à l’adresse “[http://localhost](http://localhost/)”.



*Figure 9 Vérification fonctionnement Apache2*

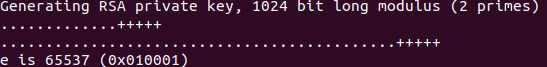
Nous constatons que la page par défaut apparaît et indique le bon fonctionnement du serveur.

## La création des certificats

### Création du certificat serveur

Afin de générer la clef privée du serveur nous effectuons la commande “openssl genrsa 1024

> fichier.key”. Cette dernière crée une clef privée sur 1024 bits et l'enregistre dans un fichier nommé “fichier.key”. En sortie, nous pouvons visualiser des informations sur la clé privée qui a été générée. Il est indiqué qu’elle est d’une taille de 1024 bits et que 𝑒 = 65537.



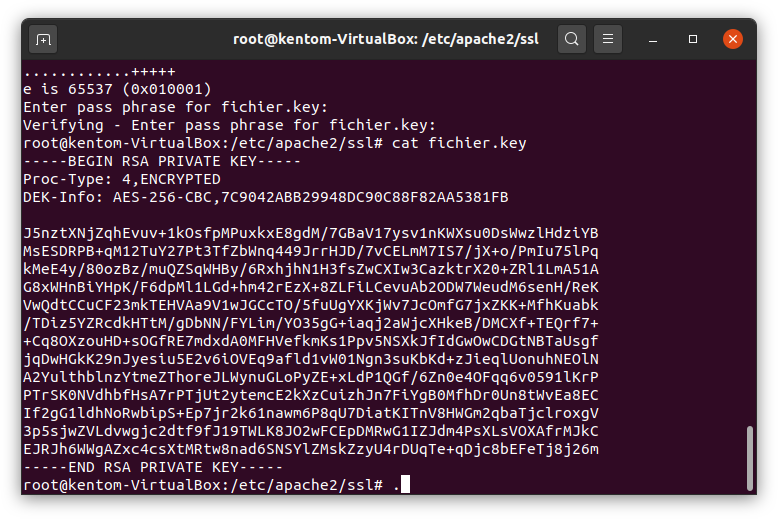
*Figure 10 Sortie génération clef privée du serveur*

Pour que cette clé soit protégée par un mot de passe il faut ajouter l’option “-aes256”. AES256 est un chiffrement symétrique. Cela donne la commande suivante “openssl genrsa -aes256 - out fichier.key 1024”. Il est ensuite demandé de saisir le mot de passe à utiliser pour protéger notre clef privée.



*Figure 11 Demande de saisir le mot de passe*

Pour afficher le contenu du fichier nous utilisons la commande “cat fichier.key”. Nous pouvons observer que le contenu est bien chiffré en AES256. Le mot de passe sert donc à déchiffrer notre clef privée.



*Figure 12 Contenu du fichier fichier.key (clef privée)*

Il est également plus prudent de restreindre l’accès au fichier uniquement aux administrateurs. Pour changer les droits sur un fichier, il faut s’assurer d’être en root et utiliser la commande “chmod 700 fichier.key”. Cette dernière permet de restreindre l’accès à ce fichier en lecture, écriture et exécution uniquement au root et de ne donner aucun droit aux utilisateurs du groupe et autres (7 = tous les droits et 0 = aucun droits).

Comme nous pouvons le voir, l’affichage du fichier pour un utilisateur normal est impossible.

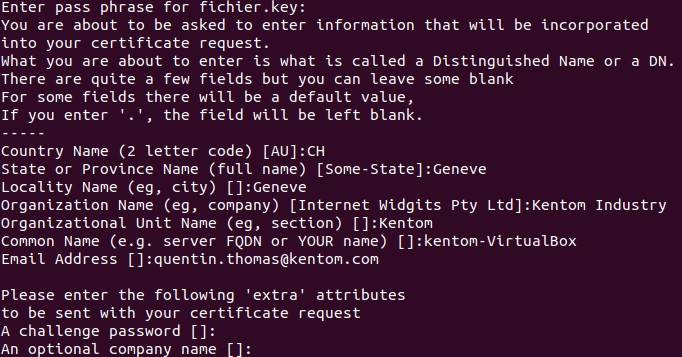


*Figure 13 Accès impossible de la clef privée, utilisateur normal*

### Création d’un fichier de demande de signature de certificat CSR

L’objectif de ce travail est de créer et signer notre propre certificat pour utiliser une connexion https (sécurisée). Pour obtenir une demande de certificat, nous entrons la commande : “openssl req -new -key fichier.key > fichier.csr”. Il est premièrement demandé de saisir le mot de passe de la clef privée.

Il faut ensuite renseigner des informations concernant le serveur web.



*Figure 14 Champs à remplir, création de la demande de certificat*

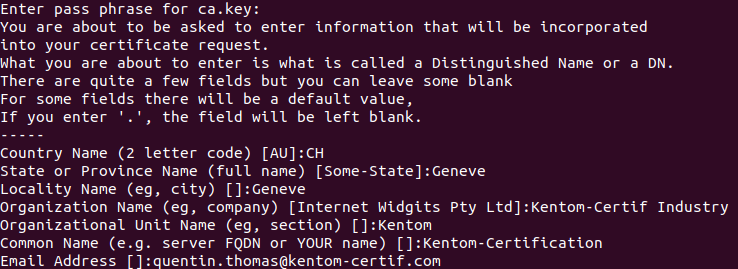
L’effet de cette commande est donc la génération d’un fichier.csr soit une demande de certificat. Cette étape est impossible à faire sans le mot de passe qui permet de déchiffrer la clef privée.

Au lieu d’auto-signer notre certificat, on aurait pu le faire signer par une autorité de certification reconnue.

### Création du certificat de l’autorité de certification

Afin de créer une clef privée pour l’autorité de certification nous réutilisons la commande “openssl genrsa -aes256 -out ca.key 1024”. Le fichier ca.key nous permet par la suite de créer le certificat de l’autorité qui est autorisé à signer des demandes de certificats.

Pour générer le certificat, on entre la commande : “openssl req -new -x509 -days 365 -key ca.key > ca.crt”. Le système nous demande de rentrer le mot de passe de la clef privée contenue dans ca.key puis entrer les informations concernant l’autorité de certification.



*Figure 15 Champs à remplir, autorité de certification*

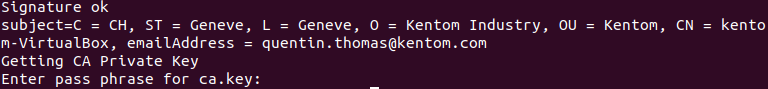
## Signature des certificats

### Signature du certificat serveur par l’autorité de certification

Pour signer la demande de certificat, on utilise la commande : “openssl x509 -req -in fichier.csr

-out fichier.crt -CA ca.crt -CAkey \ca.key -CAcreateserial -CAserial ca.srl”.

Le système demande une nouvelle fois le mot de passe pour déchiffrer la clef privée dont on a besoin pour faire la signature.



*Figure 16 Sortie signature de la demande de certificat*

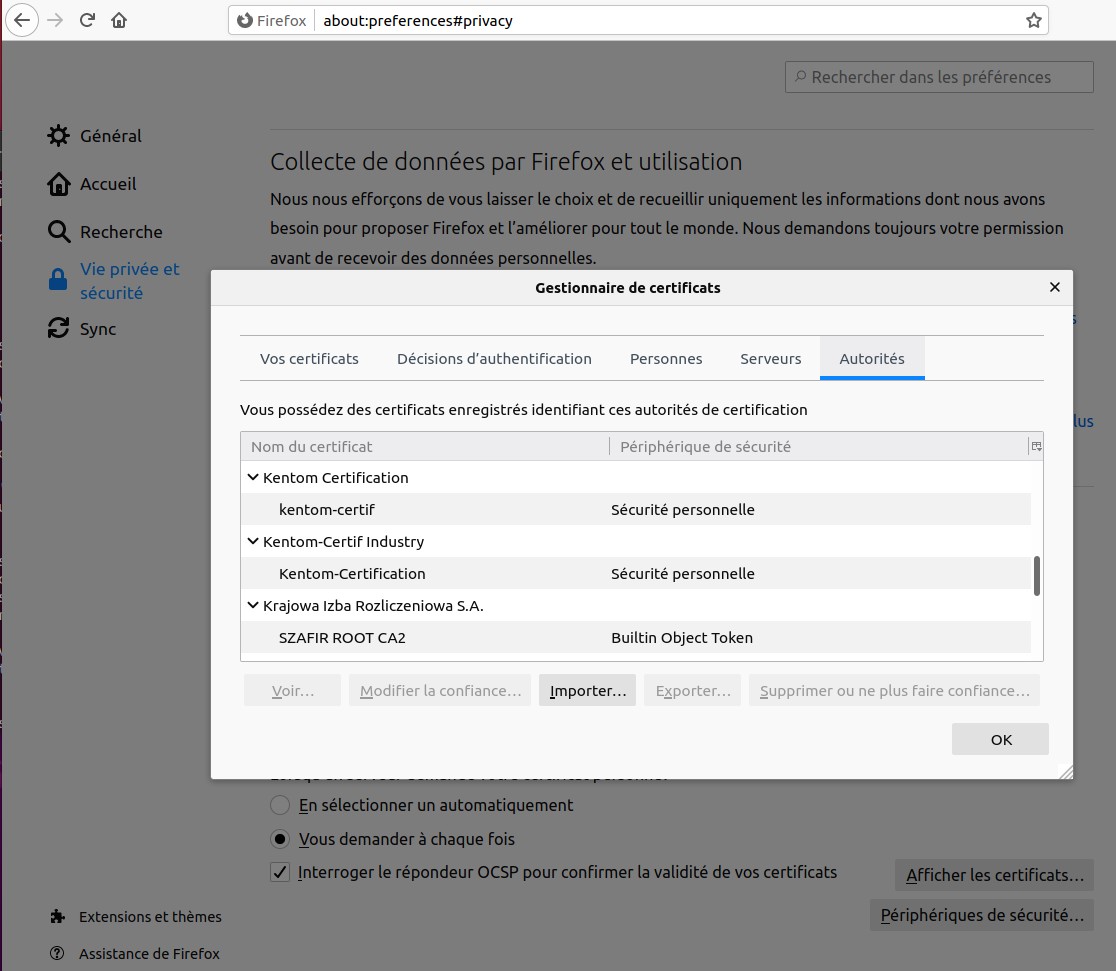
Également, l’option -CAcreateserial permet de créer un fichier.srl qui contient un numéro de série relatif au certificat de l’autorité de certification. Cette manipulation est nécessaire et uniquement possible lors de la première signature du fichier. A chaque signature, le numéro est incrémenté dans le même fichier.

Le résultat est que le fichier “ca.crt“ signe le fichier “fichier.csr”. On obtient “fichier.crt“ qui est le certificat délivré par l’autorité de certification (auto-signé).

### Installation du certificat d’autorité de certification

Le protocole utilisé est “https” soit http + ssl. C’est une connexion sécurisée.

Voici où nous nous sommes rendus sur le navigateur Mozilla Firefox afin d’ajouter un certificat.

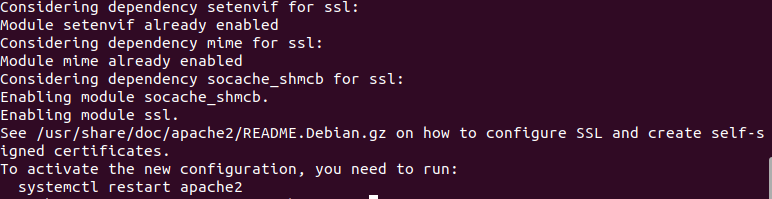


*Figure 17 Importation des certificats dans Mozilla Firefox*

Parmi les fichiers proposés à l’importation, on choisit le fichier “ca.crt“. Il s’agit du certificat de l’autorité de certification.

### Installation et configuration d’Apache2

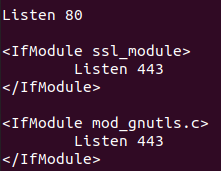
Nous savons que le module SSL est installé de base avec Apache2. Nous lançons la commande “a2enmod ssl”.



*Figure 18 Activation du module ssl*

La commande a pour effet d’activer et de configurer les modules SSL. Pour que cela prenne effet, nous utilisons la commande “systemctl restart apache2” permettant de redémarrer le service.

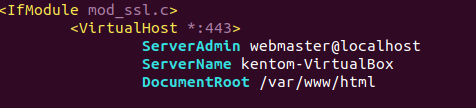
Afin de vérifier que le port 443 est en écoute nous affichons le contenu du fichier “ports.conf” situé dans “/etc/apache2/“



*Figure 19 Contenu de ports.conf*

Le numéro de port 443 correspond au protocole HTTPS et est bien en écoute car il est inscrit “listen”. Également, en écrivant la commande “netstat” on voit que des connexions existent et utilisent le protocole https.

Il faut ensuit modifier le fichier de configuration par défaut de SSL, “default-ssl.conf”. Nous avons ajouté “ServerName kentom-VirtualBox”. Comme le certificat généré est assigné au nom d’hôte kentom-VirtualBox, il faut indiquer celui-ci pour que le serveur soit valide. Il faut également, remplacer \_default\_:443 par \*:443 ce qui permet d’appliquer cette configuration ssl sur l’ensemble des hôtes et pas uniquement celui par défaut.



*Figure 20 Ajout du ServerName dans default-ssl.conf*

Nous avons aussi modifié le chemin vers le certificat et la clef privée du serveur web.



*Figure 21 Modification chemin clef privée et certificat dans default-ssl.conf*

Afin d’appliquer les modifications, nous activons le site avec la commande “a2ensite default- ssl“ puis nous redémarrons apache2 avec “systemctl restart apache2“

## Phase de test

Afin de vérifier le fonctionnement du protocole https nous nous rendons sur [https://localhost](https://localhost/) . Il est alors indiqué que le site web est non sécurisé car l’autorité de certification qui a signé le certificat n’est pas reconnue. Nous accédons quand même au site et observons que cela fonctionne.



*Figure 22 Vérification fonctionnement ssl en local*

## Conclusion

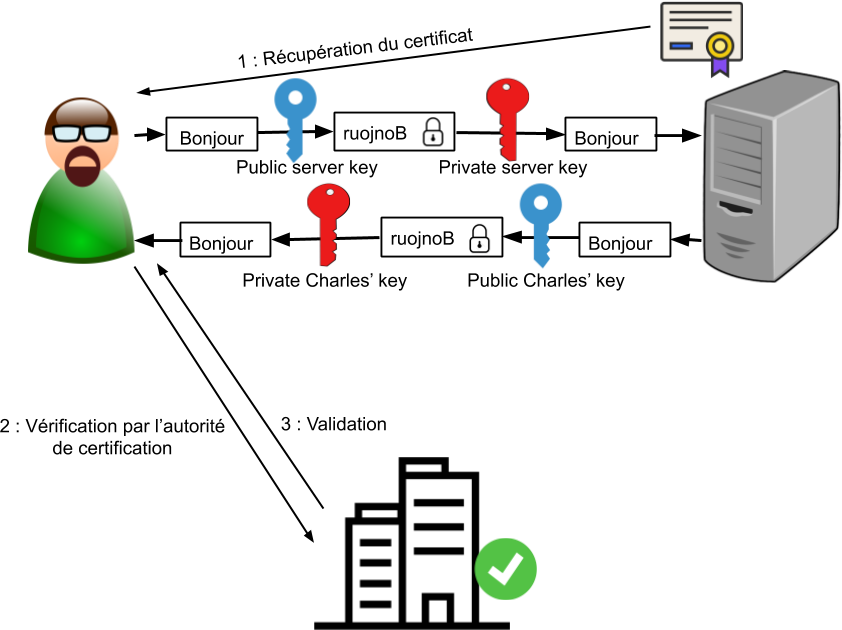
À travers ce Travail Pratique, nous remarquons donc qu’il est possible de passer par une connexion sécurisée en créant ce propre certificat. Nous sommes néanmoins et heureusement avertit que cette connexion n’est pas validée par une autorité de certificat reconnue. Cela nous fait prendre conscience qu’un site web disposant d’un certificat où il apparaît un cadenas peut être malintentionné.

Avec ce travail complexe, nous avons appris à générer des clefs RSA et compris comment fonctionne le protocole SSL. La tâche complexe a été de chercher les commandes pour répondre aux questions et finir le travail. Nous avons souvent passé du temps à comprendre comment les arguments des commandes fonctionnent tout en étant sûr qu’il s’agissait bien de la commande à utiliser.

Il nous est arrivé plusieurs fois de nous tromper avant d’acquérir les automatismes nous permettant d’aller chercher rapidement une information essentielle sur Internet ou dans le manuel disponible depuis le terminal.

# Troisième Travail Pratique : Pretty Good Privacy

## Schéma de fonctionnement



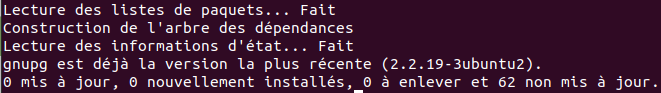
*Figure 23 Fonctionnement chiffrement asymétrique et certificat*

Le chiffrement asymétrique consiste en l’utilisation d’une clef privée et d’une clef publique. Soit Charles qui veut communiquer avec un serveur. Si Charles veut envoyer un message au serveur, il chiffre le message avec la clef publique du serveur. Le serveur peut alors déchiffrer le message avec sa clef privée que lui seul connaît. Cela permet d’avoir un échange sécurisé entre les deux machines.

Le certificat présent sur le serveur contient la clef publique ainsi que la date de validité et un tampon de l’autorité de certification (tiers de confiance). Ce dernier permet d’attester de l'authenticité de la clef publique. L’objectif est d’être sûr que la clef publique soit bien celle du serveur et non d’un espion (Man in the Middle).

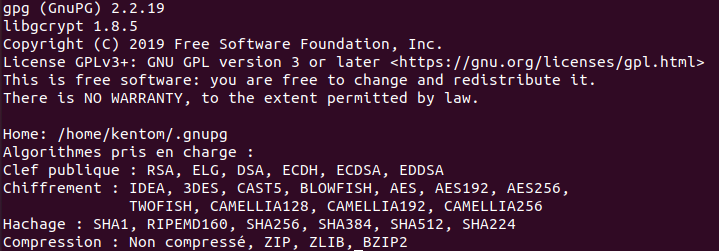
## Installation de GnuPG

Pour installer GnuPG, on utilise la commande : “ sudo apt-get install gnupg ”.



*Figure 24 Installation de GnuPG*

Une fois installé, nous avons cherché la commande permettant de savoir quels algorithmes sont pris en charge par GnuPG. Grâce au manuel, nous avons trouvé qu’il s’agit de : “gpg -- version”.



*Figure 25 Résultat de la commande gpg --version*

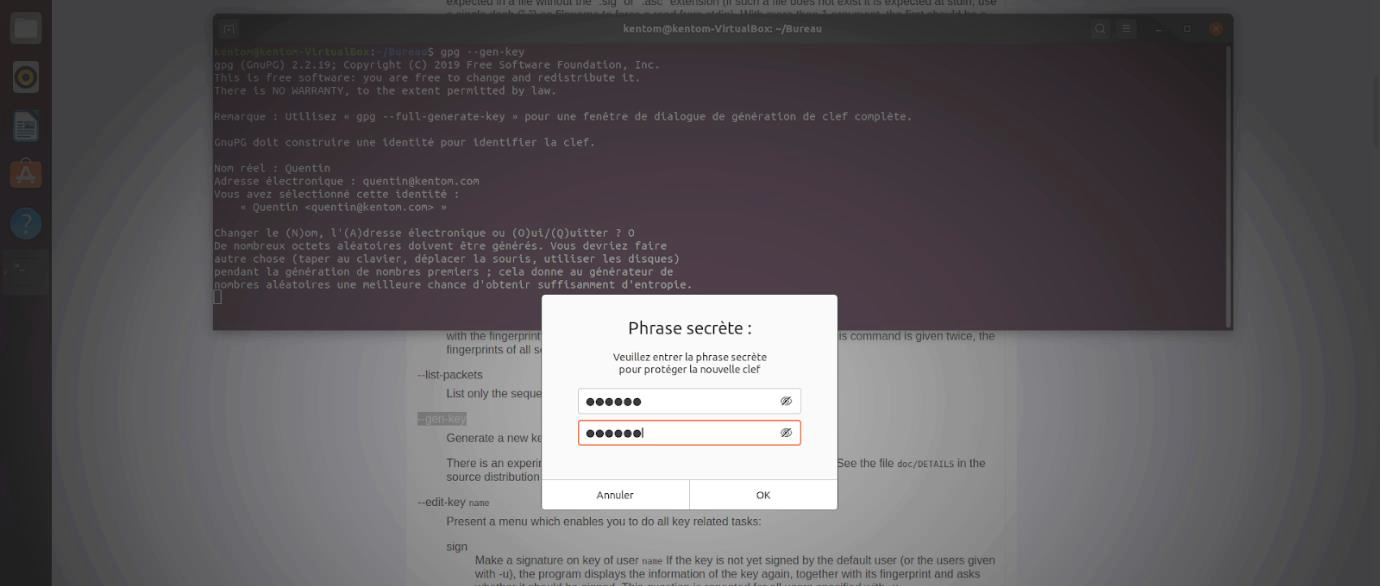
## Installation des clés GPG

À présent, nous souhaitons générer une paire de clés contenant une clef privée ainsi qu’une clef publique. Pour les générer avec les options par défaut nous utilisons la commande : “gpg

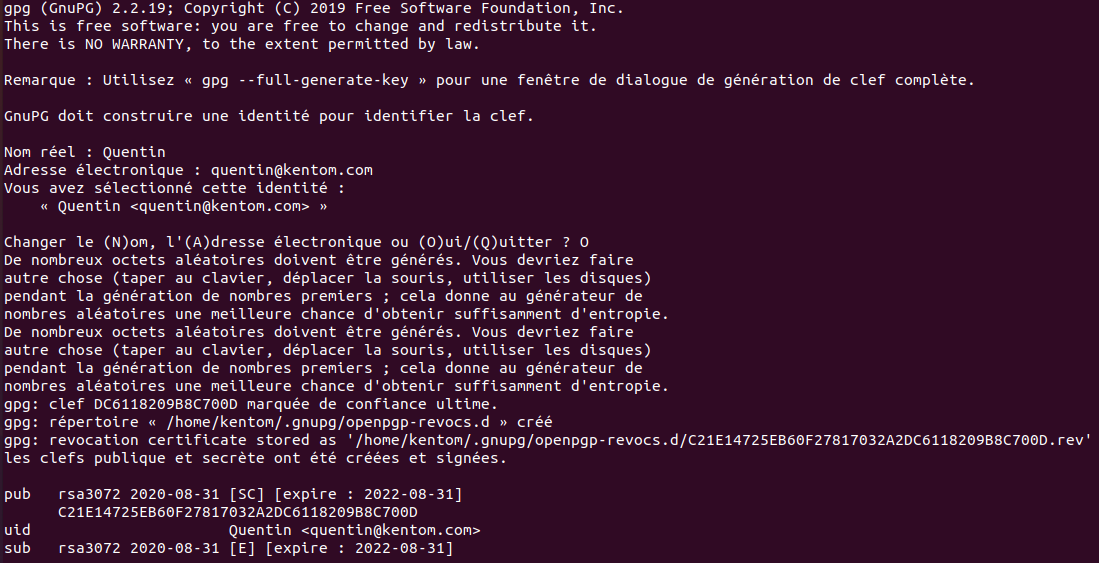
--gen-key”.

Nous devons alors renseigner des informations sur l’identité de la personne qui créer la paire de clefs dont :

* Le nom réel
* Une adresse électronique
* Un mot de passe pour protéger la paire de clef



*Figure 26 Création d'un mot de passe pour la paire de clefs*



*Figure 27 Affichage génération de la paire de clef*

Une fois les informations rentrées, des données importantes apparaissent concernant les clefs publique et privée générées. On retrouve l’identifiant qui correspond à la clef DC6118209B8C700D.

On note aussi que « pub » est la clef publique sur 3072 bits, « uid » est l’identifiant utilisateur,

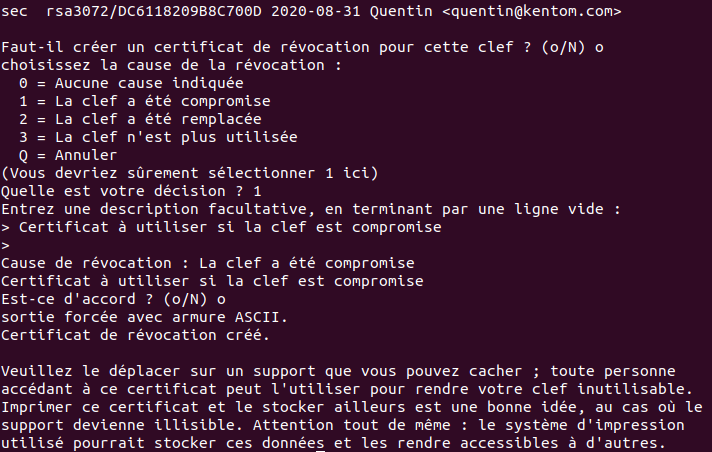
« sub » est la sous-clef publique sur 3072. Également, il est inscrit la date d’expiration pour

« pub » et « sub ».

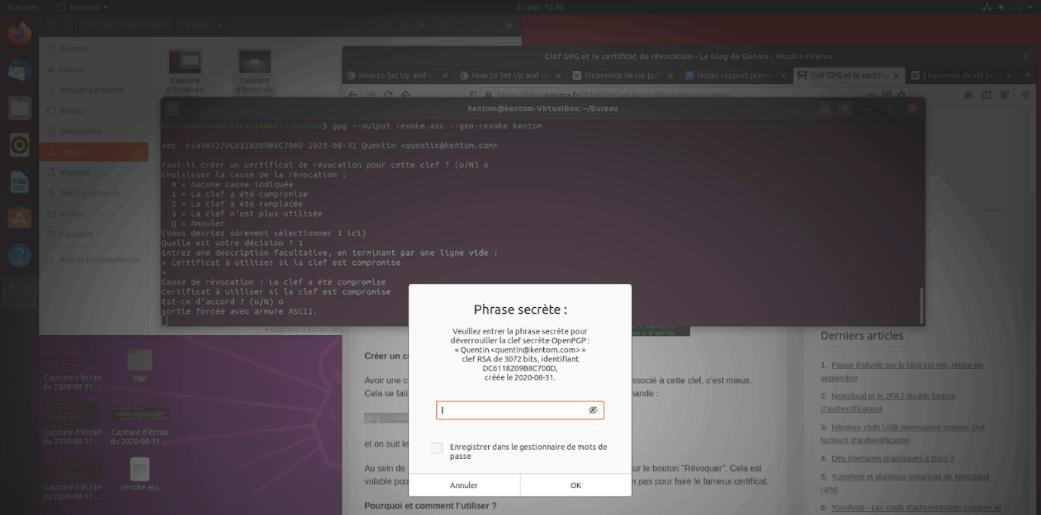
Les empreintes de clé publique sont utilisées pour simplifier certaines tâches de gestion des clefs. Elles sont plus courtes que les clés publiques correspondantes.

Un certificat de révocation permet de révoquer la clef publique. Plus personne ne peut alors l’utiliser pour chiffrer des messages. Cela permet d’annuler la clef publique lorsque la clef privée est compromise ou perdue ou que le mot de passe a été oublié. Il faut donc de préférence le générer à la création des clefs. Pour générer un certificat de révocation la clef, on utilise la commande :

“gpg --output revocation.asc --gen-revoke password\_pair\_key “



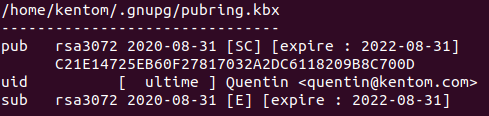
*Figure 28 Affichage génération certificat de révocation*



*Figure 29 Demande du mot de passe de ka paire de clefs*

Nous avons choisi la cause numéro 1, entré une description et renseigné le mot de passe associé aux clefs pour créer le fichier de révocation “revocation.asc“

Pour lister les clefs, on utilise la commande : “gpg --list-key “ et on observe la présence des informations relevées lors de la création de la paire de clefs. Ces dernières se trouver dans le répertoire “/home/kentom/.gnupg/pubring.kbx“



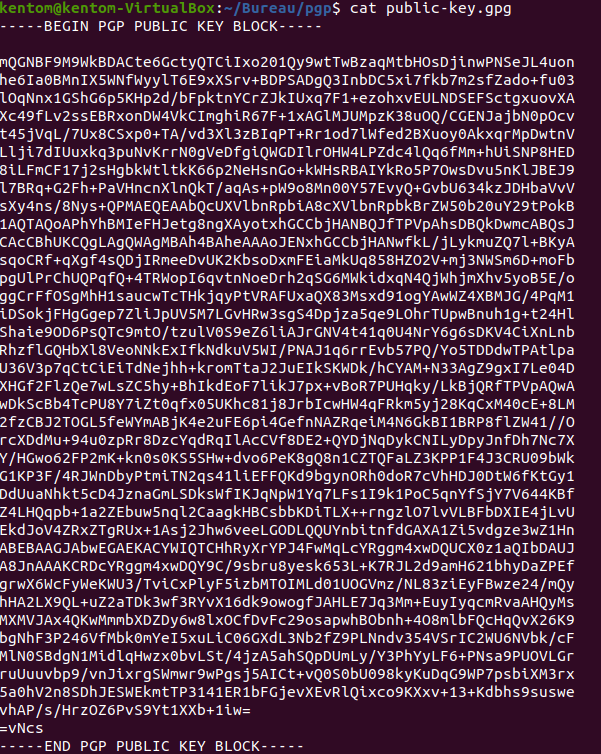
*Figure 30 Affichage des clefs, gpg --list-key*

## Exportation de la clé publique et envoie de la clé aux correspondants

Il nous est possible d’exporter la clef publique en utilisant la commande : “gpg --armor --output public-key.gpg --export [quentin@kentom.com](mailto:quentin@kentom.com)“. Il est important de donner l’adresse email associées aux clefs pour faire l’exportation. Celle-ci se fait dans le fichier “public-key.gpg“ donné en paramètre dans la commande.

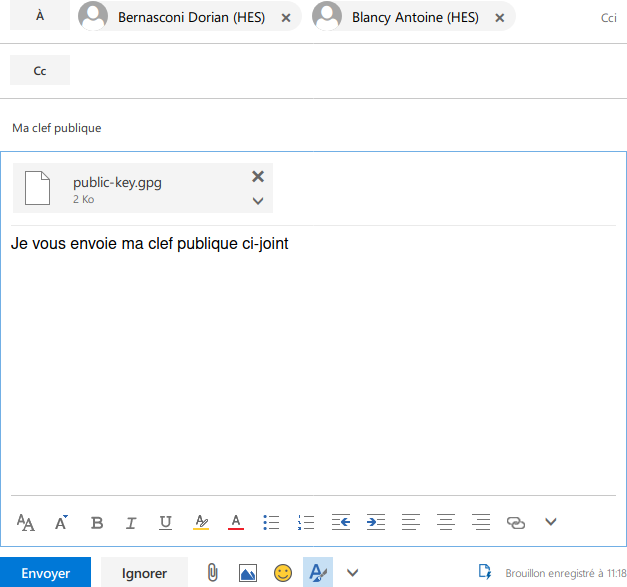
En exportant cette dernière, on remarque qu’elle est sous la forme d’un fichier texte au format ASCII. Cela est pratique car toutes les applications qui utilisent PGP peuvent lire le fichier. On peut également le transmettre à tous, très aisément par mail ou par message en prenant garde à ce que la communication soit sécurisée.

Voici le contenu du fichier “public-key.gpg“ qui contient donc notre clef publique.



*Figure 31 Contenu de public-key.gpg, clef publique*

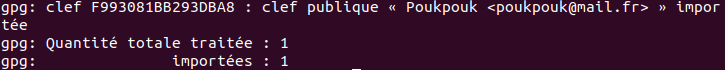
Pour envoyer la clef publique, nous nous sommes connectés à la messagerie HES et avons envoyé un mail contenant la clef publique. L’adresse de messagerie utilisée est “[quentin.rod@etu.hesge.ch](mailto:quentin.rod@etu.hesge.ch)“ et les destinataires sont Dorian Bernasconi, “[dorian.bernasconi@etu.hesge.ch](mailto:dorian.bernasconi@etu.hesge.ch)“ et Antoine Blancy, “[antoine.blancy@etu.hesge.ch](mailto:antoine.blancy@etu.hesge.ch)“. Il aurait également été possible de communiquer la clef par USB si besoin.



*Figure 32 Envoi de la clef publique par mail*

## Importation de la clef publique depuis ses correspondants pour stockage interne

Après avoir exporté la clef publique à Dorian et Antoine, nous importons également leurs clefs publiques avec la commande : “gpg --import biankey.key” et “gpg --import bernaskeyni.key”



*Figure 33 Importation de la clef publique de Dorian*

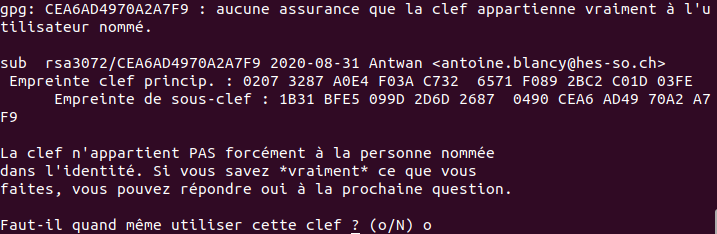
## Envoie et réception de message crypté

Nous avons créé le fichier “message.txt“ contenant “Bonjour, voici le message secret”.

Nous souhaitons envoyer ce message à Antoine et nous utilisons donc sa clef publique pour chiffrer le message. La commande utilisée est: “gpg --output message\_crypted --encrypt -- recipient [antoine.blancy@hes-so.ch](mailto:antoine.blancy@hes-so.ch) message.txt”.

Il est alors demandé d’entrer le mot de passe associé à nos clefs.

Après avoir entré le nom de l’utilisateur, un message apparaît nous indiquant que la clef publique n’est pas forcément celle d’Antoine. La cause de ce message est que la clef d’Antoine n’est pas signée donc pas reconnue et pourrait provenir d’une usurpation.

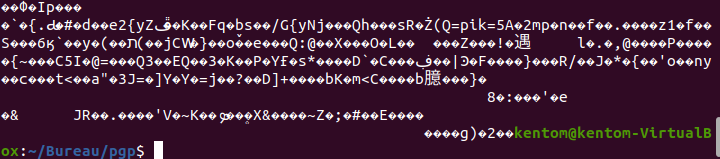


*Figure 34 Chiffrement d'un message avec la clef publique d'Antoine*

Pour s’assurer que la clef donnée est bien celle d’Antoine, on doit demander l'empreinte et recevoir cette dernière d’une autre manière fiable (clef USB...). Le seul moyen de savoir que la clef publique est bonne, c’est de comparer les deux empreintes.

On observe aussi la présence de l'identifiant de la clef publique, sa taille, la date d’expiration et les empreintes de la clef principale et de la sous-clef. C’est avec ces informations que l’on est capable de vérifier que la clef publique est bonne.

Le résultat de cette commande est la création du fichier : “message\_crypted.asc“. Dont le contenu est affiché ci-dessous :



*Figure 35 Contenu de message\_crypted.asc, message chiffré*

Il s’agit d’un fichier binaire qui pèse 499 octets.

Le fichier non chiffré est plus lourd et pèse 740 octets.

Nous avons ensuite envoyé par mail ce fichier et Antoine nous a transmis le sien. Il se nomme “message\_antoine“

Pour décrypter le message envoyé par Antoine, on utilise la commande : “gpg --output message\_antoine\_decrypted.txt --decrypt message\_antoine” avec message\_antoine, le message chiffré.

La phase de déchiffrement se fait en utilisant notre clef privée. De ce fait, on nous demande de renseigner le mot de passe protégeant la clef privée pour éviter que n’importe qui ne puisse déchiffrer le message secret.

En affichant “message\_antoine\_decrypted.txt“ on obtient de manière lisible le message envoyé par Antoine :



*Figure 36 Contenu du message d'Antoine déchiffré*

Si on envoie un message crypté et qu’il ne possède pas notre clef publique, la communication ne peut se faire qu’entre nous et le destinataire. Il peut déchiffrer ce qu’on lui envoie mais lui ne peut pas crypter pour nous envoyer le message.

## Création de la signature

La signature d’un fichier permet de valider l’authenticité de l’émetteur.

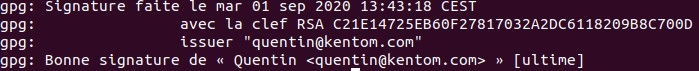
Afin d’obtenir un fichier contenant à la fois les données du fichier signé et la signature, on utilise la commande : “ gpg --sign --defalut-key [quentin@kentom.com](mailto:quentin@kentom.com) message.txt”. Ici, le fichier à signer est ”message.txt” et le fichier signé est au format “gpg“.

Si on souhaite séparer la signature du fichier que l’on veut signer, on utilise la commande : “ gpg -sb message.txt”. Dans ce cas, on nous demande de rentrer le mot de passe pour accéder à la clef privée dont on a besoin pour signer le fichier. Le format de ce dernier contenant uniquement la signature est “sig“ .

Pour observer les algorithmes, on peut utiliser la commande : “ gpg --clearsign message.txt”. Un fichier est alors créé au format “asc“. On observe que le contenu est hashé avec l’algorithme SHA512. Ceci permet de vérifier que le contenu du message n’a pas été modifié après la signature. Sinon, le hash n’est plus du tout le même.

## Vérification de la signature

Pour vérifier une signature, on peut utiliser la commande : “gpg --verify message\_signed” ainsi, on utilise la clef publique de la personne qui a signé pour effectuer la vérification.



*Figure 37 Vérification de la signature*

Si un message est édité après signature, le document ne sera plus authentique.

En effet, comme la signature dépend du contenu du document, la signature ne vérifiera plus le bon document qui ne pourra, par la suite, plus être déchiffré.

## Questions complémentaires

Pour crypter en PGP, la configuration minimale requise est de posséder une paire de clefs, on n’a pas forcément besoin d’un mot de passe mais il est très conseillé pour protéger sa clef privée.

Si on envoie un mail crypté à un destinataire n’utilisant pas OpenPGP, il lui sera possible de le déchiffrer car tous les logiciels utilisant PGP sont compatibles entre eux. La paire de clefs dépend uniquement de l’utilisateur donc est utilisable sur n’importe quel ordinateur.

Si on utilise une connexion type SSL ou SSH, cela permet d’apporter une sécurité supplémentaire lors des communications. Ainsi, toutes les données sont chiffrées et ne peuvent pas être interceptées.

PGP est un moyen de chiffrement Open Source et disponible entre toutes les plateformes ce qui n’est pas le cas des autres solutions de cryptage.

## Conclusion

En somme, ce travail pratique nous a permis de comprendre le fonctionnement du chiffrement en utilisant des clefs publiques et privées. Au travers d’exemples et d’échange de messages chiffrés, nous avons à chaque étape compris ce qui se passait et pourquoi cela se faisait ainsi.

Contrairement au second travail pratique, nous n’avons pas rencontré de difficultés majeur dans la réalisation du travail demandé.

Les connaissances que nous avons acquises au cours de ce travail nous permettent aussi de compléter les cours sur le chiffrement RSA que nous avions eu cette année en mathématiques et en sécurité.

# Quatrième Travail Pratique : Firewall & Services

## Firewall sous CentOS 7

Pour débuter, les machines de l’école étant sur Ubuntu, il est nécessaire d’installer une machine virtuelle CentOS 7 sur ces dernières pour pouvoir démarrer le travail pratique.

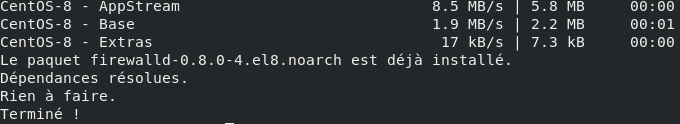
### Installation de firewalld

L’installation est possible uniquement en mode Administrateur. Nous commençons donc par lancer l’invite de commande et entrons dans le mode root avec la commande “sudo su root”. Pour vérifier que Firewalld est installé, on peut utiliser la commande : “whereis firewalld”



*Figure 38 Vérification existence firewalld*

Cette commande nous retourne le chemin vers un fichier dont le nom est firewalld au format “gz“. Cette extension désignant des fichiers compressés, nous en déduisons que firewalld n’est pas installé par défaut. Pour ce faire, on utilise la commande : “yum install firewalld”



*Figure 39 Installation de firewalld*

Il faut alors désactiver firewalld. Pour cela, nous effectuons la commande “systemctl disable firewalld”.



*Figure 40 Désactivation de firewalld*

Par la suite, nous devons relancer le service firewalld. Pour cela, nous utilisons la commande: “systemctl restart firewalld”

### Observation des zones

Une zone permet de définir un groupe d’ordinateurs. Il est alors possible d’appliquer des restrictions sur ce dernier. Nous pouvons alors définir les protocoles autorisés mais aussi les accès entre différentes zones. Par exemple, une zone Informatique et une seconde Administration pourraient avoir toutes deux le droit d’aller sur Internet mais l’interdiction de communiquer entre elles.

Pour affecter une machine à la zone souhaitée, il suffit d’ajouter l’interface réseau dans la zone. Il existe une zone par défaut qui contient l’ensemble des interfaces réseaux qui ne sont dans aucune autre zone.

Pour déterminer les zones disponibles dans firewalld, on utilise la commande : “firewall-cmd -

-get-zones”:



*Figure 41 Zones disponibles à l'installation*

Il existe également la commande : “firewalld-cmd --get-default-zone” qui permet de connaitre la zone sélectionnée par défaut : 

Pour identifier les différences entre les deux dossiers zones, nous affichons le contenu de chacun d’eux. Pour le dossier dont le chemin d’accès est “/usr/lib/firewalld/zones” le résultat est :



*Figure 42 fichiers de configuration des zones dans /usr/lib/firewalld/zones*

En ce qui concerne le dossier du répertoire “/etc/firewalld/zones”, le résultat est :

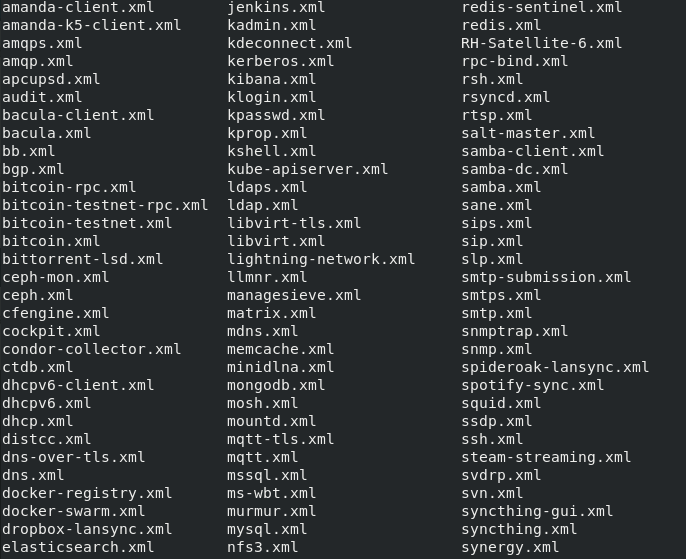


*Figure 43 Fichiers de configuration des zones dans /etc/firewalld/zones*

On constate que le fichier “public.xml“ est présent dans les deux dossiers. En revanche, le dossier “/usr/lib/firewalld/zones” contient le fichier de configuration de toutes les zones existantes de base à l’installation. On en déduit que le dossier “/etc/firewalld/zones” contient les configurations activées.

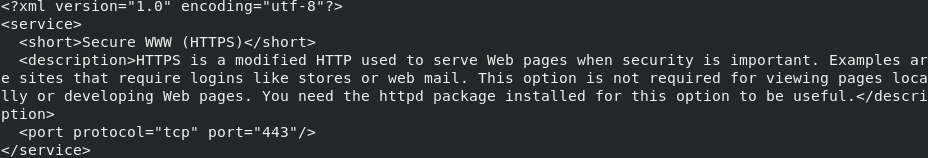
### Observation des fichiers de services

Pour voir les services disponibles avec firewalld, on peut se diriger vers le dossier “/usr/lib/firewalld/services” et afficher le contenu du dossier.



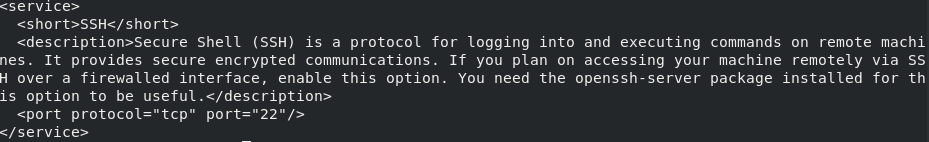
*Figure 44 Fichier de configuration des services /usr/lib/firewalld/services*

Pour comprendre à quoi servent les fichiers de service, nous en ouvrons un et remarquons qu’il est possible de configurer le protocole utilisé ainsi que le port attribué pour le service. Ce fichier est au format XML et existe pour chaque service.



*Figure 45 Fichier de configuration de https.xml*

En lisant le contenu du dossier, nous trouvons le fichier “/usr/lib/firewalld/services/ssh.xml” que nous affichons avec la commande : “cat ssh.xml"



*Figure 46 fichier de configuration de ssl.xml*

Comme nous pouvons voir, ce fichier contient une description du service ainsi que le port utilisé et enfin le protocole de la couche transport.

Afin de visualiser les services déjà initialisés par firewalld nous utilisons la commande “firewall

-cmd --list-services”.



### Création d’une nouvelle zone

*Figure 47 Services initialisés*

Pour créer une nouvelle zone, on utilise la commande : “firewall-cmd --permanent --new- zone=SSIzone”. Cette commande nous retourne la mention “success”. Il faut alors recharger firewalld avec la commande : “firewall-cmd --reload”. Une fois cette opération effectuée, nous pouvons vérifier avec la commande : “firewall-cmd --get-zones” que la zone a bien été créée. En effet, elle est inscrite dans la liste ci-dessous.



*Figure 48 Affichage des zones existantes*

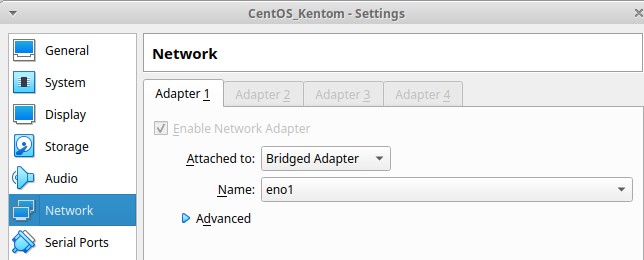
Il est aussi possible de trouver son fichier de configuration qui se situe dans le répertoire : ”/etc/firewalld/zones”. Son contenu et pour l’instant vierge :



*Figure 49 Contenu fichier SSIzone.xml, création*

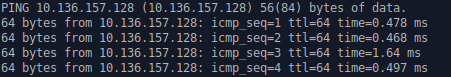
### Configuration de la zone SSIzone

Avant de continuer le travail pratique, nous essayons de faire un ping depuis un ordinateur de l’école vers la machine virtuelle situé sur un autre ordinateur de l’école.

Malheureusement, cette commande ne fonctionne pas car la machine virtuelle n’est pas sur le bon réseau. La communication est alors impossible. Nous avons alors passé du temps à trouver comment résoudre ce problème. La solution réside dans la mise en place d’un “bridged Adapter“ sur la machine virtuelle.

*Figure 50 Bridge Adaptator VirtualBox*

Après cette manipulation, le ping est désormais possible.



*Figure 51 Ping fonctionnel Thomas -> Quentin*

Ensuite, nous devons modifier notre fichier de zone créé auparavant pour refuser toutes les connections entrantes.

Pour cela, nous devons utiliser la commande : “firewall-cmd --permanent --zone=SSIzone -- set-target=REJECT”. Le paramètre “--permanent” est indispensable pour l’utilisation de cette commande.

Comme aucune autorisation n’est accordée parmi les différentes connexions possibles, elles sont toutes rejetées. Pour appliquer cette nouvelle configuration, il faut recharger firewalld avec la commande: “firewall-cmd --reload”

### Activation de la zone

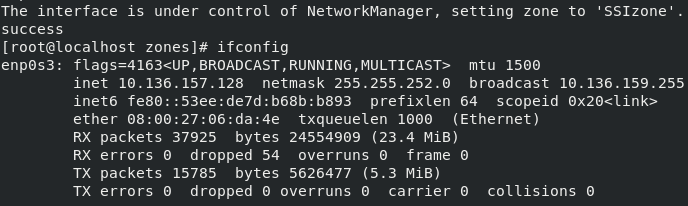
Pour voir les modifications apportées à la zone, on peut afficher le contenu du fichier “SSIzone.xml” avec la commande « cat SSIzone.xml ». On observe alors que les modifications ont bien été appliquées au fichier.



*Figure 52 Contenu fichier SSizone.xml, ajout de REJECT*

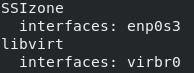
Par la suite, nous devons modifier la zone par défaut et la remplacer par la zone que nous avons créée. Pour cela, on utilise la commande : “firewall-cmd --set-default-zone *SSIzone”.*

Il faut aussi assigner l’interface par défaut à la zone. En utilisant la commande “ifconfig”, on observe que l’interface par défaut se nomme : “enp0s3”. Ainsi, on peut utiliser la commande : “firewall-cmd --permanent --zone=SSIzone --add-interface=enp0s3”. Suite à l'exécution, on nous annonce que la manipulation a été exécutée avec succès.



*Figure 53 Résultat de la commande IFCONFIG sur la VM de Quentin*

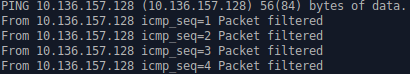
Nous pouvons vérifier que l’interface est bien dans la zone en utilisant la commande : “firewall- cmd --get-active-zones”. On remarque que la zone active est bien celle que l’on a créée et qu’elle est bien reliée à l’interface par défaut “enp0s3”.



*Figure 54 Interfaces et zones actives*

### Test de la zone

Dès lors, si nous essayons d’effectuer un ping vers la machine virtuelle, nous obtenons ce résultat :



*Figure 55 Ping non fonctionnel Thomas -> Quentin*

Ceci nous indique que la commande a permis d’atteindre la machine virtuelle sans pour autant échanger avec elle. Cela à cause des restrictions appliquées sur la zone SSIzone dans laquelle figure l’interface par défaut.

## VSFTPD sous CentOS 7

### Installation de VSFTPD (Very Secure File Transfer Protocol Daemon)

La première étape consiste à installer le paquet VSFTPD. Pour faire ceci, on rentre dans le terminal la commande: ”yum -y install vsftpd”.



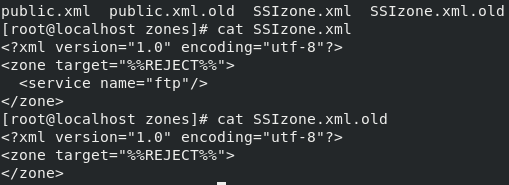
*Figure 56 Installation vsftpd*

Le service est par défaut désactivé. Il faut l’activer manuellement en utilisant la commande : “systemctl start vsftpd”.

### Ouverture des ports pour l’utilisation de FTP

Pour autoriser l’utilisation de FTP, il est indispensable d’autoriser le port 21 qui correspond à ce protocole. Nous l’ajoutons donc dans le fichier “SSIzone.xml“. Pour ce faire, nous utilisons la commande : “firewall-cmd --permanent --zone=SSIzone --add-service=ftp”.

Il faut néanmoins recharger firewalld pour prendre en compte les changements avec la commande : “ firewall-cmd --reload”. Nous affichons le fichier de configuration pour s’assurer des changements.



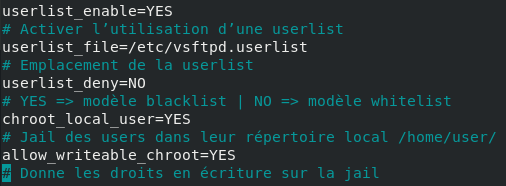
*Figure 57 Contenu fichier SSIzone, autorisation FTP*

On observe que le fichier SSIzone.xml a été modifié comme nous le souhaitions. De ce fait, un fichier SSIzone.xml.old a été créé et contient la dernière configuration avant la modification.

### Configuration du serveur FTP

Pour sauver le contenu du fichier “vsftpd.conf”, il suffit de copier les données dans un nouveau fichier avec la commande : “cp vsftpd.conf vsftpd\_save.conf”. La sauvegarde est donc écrite dans le fichier “vsftpd\_save.conf“

Nous avons ajouté les lignes de la façon suivante dans le fichier vsftpd.conf:



*Figure 58 Lignes ajoutées dans vsftpd.conf*

Comme nous pouvons le remarquer, il faut impérativement écrire chaque commentaire sur une ligne respective. Sinon, une erreur se produit et il est impossible de démarrer le service.

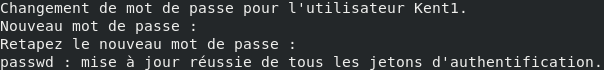
Une fois le fichier modifié et les informations ajoutées nous devons entrer la commande : “semanage boolean -m ftpd\_full\_access --on”.

Selinux permet de restreindre l’accès de fichiers ou d’actions à certains programmes / protocoles. Des booléens existent pour contrôler le type d’autorisation. Ici, grâce à cette commande, le protocole ftpd a tous les droits et accès sur la machine.

### Préparation de l’accès au service FTP (À faire par 2 obligatoirement)

Cette partie devant se faire avec deux ordinateurs configurés dans une zone SSIzone, comme aucun autre groupe n’était à cette étape nous avons recréer une seconde VM. Cette dernière a été configurée de manière identique.

Pour la suite du travail pratique, il faut créer sur chacun des ordinateurs un autre utilisateur. Nous devons d’abord ajouter le nouvel utilisateur de Quentin sur l’ordinateur de Thomas avec la commande “sudo adduser Kent1”. Ensuite nous devons paramétrer un mot de passe pour l’utilisateur Kent1 avec la commande : “sudo passwd Kent1”.



*Figure 59 Création utilisateur Kent1 sur VM de Thomas*

Nous avons effectué les mêmes opérations sur la machine de Quentin en créant un utilisateur “2hmas” (two-hmas).

Nous devons aussi créer le fichier vsftpd.userlist sur les deux machines pour lister les utilisateurs autorisés. Il doit donc figurer Kent1 sur la liste de l’ordinateur de Thomas et 2hmas sur celle de la machine de Quentin.

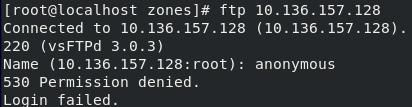
Sur la machine de Quentin, il est saisi la commande “echo "2hmas" | tee -a /etc/vsftpd.userlist” et sur celle de Thomas, “echo "Kent1" | tee -a /etc/vsftpd.userlist

Le mot-clef “tee” permet de lire l’input “Kent1” et de l’écrire dans le fichier “vsftpd.userlist” créé grâce au pipe présent dans la commande. On va aussi afficher le résultat dans le terminal une fois l'opération effectuée avec le mot-clef : “echo”.

### Test de l’accès au service FTP du binôme

Il faut tout d’abord installer le paquet FTP avec la commande : “yum install ftp”

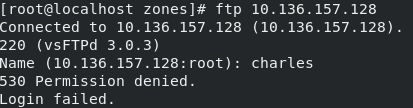
Nous nous connectons avec le PC de Thomas sur la machine de Quentin. Pour cela, nous utilisons la commande “ftp 10.136.157.128”. L’adresse IP 10.136.157.128 est celle de la VM de Quentin.



*Figure 60 Connexion FTP sur VM Quentin avec Anonymous*

La connexion avec l’utilisateur “Anonymous“ est impossible car dans le fichier “vsftpd.conf” nous avons indiqué anonymous\_enable=NO. Cela permet d’empêcher une connexion en tant qu’anonyme (Anonymous). Il s’agit d’un utilisateur qui ne nécessite pas de mot de passe pour se connecter.

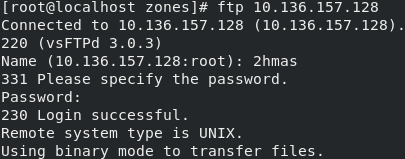
Ensuite, nous essayons de nous connecter en utilisant l’utilisateur “Charles” présent sur le PC de Thomas.



*Figure 61 Connexion FTP sur VM Quentin avec Charles*

Comme nous pouvons constater, il est impossible de se connecter. En effet, uniquement un compte sur la machine virtuelle de Quentin est utilisable.

Enfin nous essayons en utilisant l’utilisateur “2hmas” créé sur l’ordinateur de Quentin.



*Figure 62 Connexion FTP sur VM Quentin avec 2hmas*

Cette fois-ci, la connexion est possible.

L’option : “allow\_writeable\_chroot=YES” autorise le fait de modifier le dossier root par l'utilisateur FTP. Cette commande est déconseillée car un utilisateur malveillant peut modifier ou activer certains services en passant par le protocole FTP et ceci, à cause de cette option.

### Sécurisation supplémentaire

Afin de réduire l’accès à l’utilisateur FTP nous avons mis en commentaire la ligne “allow\_writeable\_chroot=YES” du fichier de configuration “vsftpd.conf”.

Ensuite, nous créons le dossier ftp dans un dossier qui est uniquement accessible par l’utilisateur, “/home/2hmas”. Pour cela, nous utilisons la commande “mkdir ftp“.

Il faut alors donner le minimum de droits possibles à tous les utilisateurs sur le fichier “ftp”. Pour cela, on utilise la commande : “chown nobody:nobody ftp”. Nobody est un utilisateur qui a le moins de droits possibles et qui ne possède aucun fichier.

Afin d’enlever les droits en écritures sur l’ensemble des fichiers contenus dans “ftp”, on utilise la commande : “chmod -w ftp”.

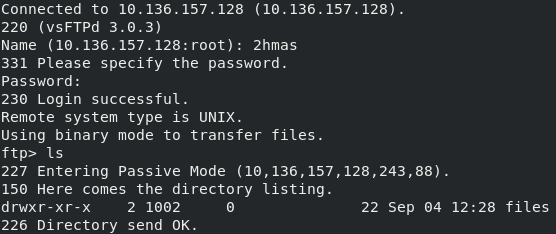
Par la suite, nous voulons créer un dossier files dans lequel Thomas pourra transférer des fichiers depuis sa VM sur la VM de Quentin. Après avoir créé ce dossier, nous devons l’associer à un utilisateur. Ici, le seul que l’on souhaite autoriser est 2hmas donc on le nomme propriétaire du dossier “files” avec la commande : “chown 2hmas files”.

De plus, il faut que le propriétaire ait le droit d’écrire dans ce dossier sachant qu’il peut déjà lire et exécuter, ce qui n’est pour l’instant pas possible avec la commande chown, utilisée sur le dossier ftp et tout son contenu. Il faut donc donner les droits en écriture au propriétaire avec

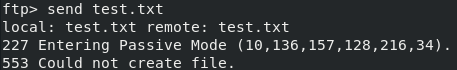
la commande : “chmod 755 files”. Le 7 indique que le propriétaire a tous les droits, les deux 5 consécutifs indiquent que le groupe et autres usagers conservent uniquement leurs droits de lecture et exécution.

Avant de pouvoir tester la connexion ftp, il reste à vérifier l’identité de l’utilisateur qui se connecte avec ftp pour pouvoir le rediriger. Nous devons donc modifier le fichier “/etc/vsftp/vsftpd.conf”. La modification va de ce fait impacter le répertoire dans lequel l’ensemble des utilisateurs vont arriver une fois la connexion établie.

Une fois toute la configuration faite, nous redémarrons le service vsftpd avec la commande “systemctl stop vsftpd“ puis “systemctl start vsftpd“. Nous testons ensuite de nous connecter en FTP depuis la VM de Thomas sur celle de Quentin avec le compte 2hmas.

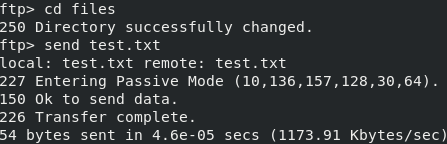


*Figure 63 Résultat de la commande ls avec l’utilisateur 2hmas*



*Figure 64 Envoi d'un fichier sur VM de Quentin dans le dossier ftp*

Nous nous déplaçons donc dans le dossier : “files” et observons qu’il est, cette fois-ci, possible d’envoyer un fichier depuis la VM de Thomas vers la VM de Quentin en passant par le protocole FTP. Pour cela, nous avons utilisé la commande “send” sur un fichier existant et créé sur la VM de Thomas.



*Figure 65 Envoi d'un fichier dans le dossier files*

## Conclusion

Au cours de ce travail pratique, nous avons pu mettre en place un par feu qui est un élément de sécurité essentiel tant pour les particuliers que dans le milieu professionnel. Nous avons appris à créer des zones, ajouter des machines dans ces dernières et aussi comment autoriser uniquement certains protocoles. Dans un second temps, nous avons mis en place une connexion FTP pour un utilisateur précis. Il a alors fallu créer un dossier réservé à l’utilisateur et attribuer correctement les droits afin d’avoir un système sécurisé et répondant aux attentes.

# Table des illustrations

[Figure 1 Fonctionnement global du programme 4](#_bookmark3)

[Figure 2 Opérations lors de la lecture d'un fichier texte 6](#_bookmark8)

[Figure 3 Correspondance entre les clefs de chiffrements et de déchiffrements 7](#_bookmark12)

[Figure 4 Operations lors de l'écriture dans un fichier texte 9](#_bookmark17)

[Figure 5 Exécution du code et affichage du terminal 10](#_bookmark19)

[Figure 6 Comparaison fichier initial et résultat 10](#_bookmark20)

[Figure 7 Vérification des différences entre les deux fichiers 10](#_bookmark21)

[Figure 8 Installation Apache2 11](#_bookmark25)

[Figure 9 Vérification fonctionnement Apache2 11](#_bookmark26)

[Figure 10 Sortie génération clef privée du serveur 12](#_bookmark29)

[Figure 11 Demande de saisir le mot de passe 12](#_bookmark30)

[Figure 12 Contenu du fichier fichier.key (clef privée) 12](#_bookmark31)

[Figure 13 Accès impossible de la clef privée, utilisateur normal 13](#_bookmark32)

[Figure 14 Champs à remplir, création de la demande de certificat 13](#_bookmark34)

[Figure 15 Champs à remplir, autorité de certification 14](#_bookmark36)

[Figure 16 Sortie signature de la demande de certificat 14](#_bookmark39)

[Figure 17 Importation des certificats dans Mozilla Firefox 15](#_bookmark41)

[Figure 18 Activation du module ssl 15](#_bookmark43)

[Figure 19 Contenu de ports.conf 16](#_bookmark44)

[Figure 20 Ajout du ServerName dans default-ssl.conf 16](#_bookmark45)

[Figure 21 Modification chemin clef privée et certificat dans default-ssl.conf 16](#_bookmark46)

[Figure 22 Vérification fonctionnement ssl en local 17](#_bookmark48)

[Figure 23 Fonctionnement chiffrement asymétrique et certificat 18](#_bookmark52)

[Figure 24 Installation de GnuPG 19](#_bookmark54)

[Figure 25 Résultat de la commande gpg --version 19](#_bookmark55)

[Figure 26 Création d'un mot de passe pour la paire de clefs 20](#_bookmark57)

[Figure 27 Affichage génération de la paire de clef 20](#_bookmark58)

[Figure 28 Affichage génération certificat de révocation 21](#_bookmark59)

[Figure 29 Demande du mot de passe de ka paire de clefs 21](#_bookmark60)

[Figure 30 Affichage des clefs, gpg --list-key 21](#_bookmark61)

[Figure 31 Contenu de public-key.gpg, clef publique 22](#_bookmark63)

[Figure 32 Envoi de la clef publique par mail 23](#_bookmark64)

[Figure 33 Importation de la clef publique de Dorian 23](#_bookmark66)

[Figure 34 Chiffrement d'un message avec la clef publique d'Antoine 24](#_bookmark68)

[Figure 35 Contenu de message\_crypted.asc, message chiffré 24](#_bookmark69)

[Figure 36 Contenu du message d'Antoine déchiffré 25](#_bookmark70)

[Figure 37 Vérification de la signature 25](#_bookmark73)

[Figure 38 Vérification existence firewalld 27](#_bookmark79)

[Figure 39 Installation de firewalld 27](#_bookmark80)

[Figure 40 Désactivation de firewalld 27](#_bookmark81)

[Figure 41 Zones disponibles à l'installation 28](#_bookmark83)

[Figure 4 fichiers de configuration des zones dans /usr/lib/firewalld/zones 28](#_bookmark84)

[Figure 43 Fichiers de configuration des zones dans /etc/firewalld/zones 28](#_bookmark85)

[Figure 44 Fichier de configuration des services /usr/lib/firewalld/services 29](#_bookmark87)

[Figure 45 Fichier de configuration de https.xml 29](#_bookmark88)

[Figure 46 fichier de configuration de ssl.xml 29](#_bookmark89)

[Figure 47 Services initialisés 30](#_bookmark91)

[Figure 48 Affichage des zones existantes 30](#_bookmark92)

[Figure 49 Contenu fichier SSIzone.xml, création 30](#_bookmark93)

[Figure 50 Bridge Adaptator VirtualBox 30](#_bookmark95)

[Figure 51 Ping fonctionnel Thomas -> Quentin 31](#_bookmark96)

[Figure 52 Contenu fichier SSizone.xml, ajout de REJECT 31](#_bookmark98)

[Figure 53 Résultat de la commande IFCONFIG sur la VM de Quentin 31](#_bookmark99)

[Figure 54 Interfaces et zones actives 32](#_bookmark100)

[Figure 55 Ping non fonctionnel Thomas -> Quentin 32](#_bookmark102)

[Figure 56 Installation vsftpd 32](#_bookmark105)

[Figure 57 Contenu fichier SSIzone, autorisation FTP 33](#_bookmark107)

[Figure 58 Lignes ajoutées dans vsftpd.conf 33](#_bookmark109)

[Figure 59 Création utilisateur Kent1 sur VM de Thomas 34](#_bookmark111)

[Figure 60 Connexion FTP sur VM Quentin avec Anonymous 34](#_bookmark113)

[Figure 61 Connexion FTP sur VM Quentin avec Charles 35](#_bookmark114)

[Figure 62 Connexion FTP sur VM Quentin avec 2hmas 35](#_bookmark115)

[Figure 63 Résultat de la commande ls avec l’utilisateur 2hmas 36](#_bookmark117)

[Figure 64 Envoi d'un fichier sur VM de Quentin dans le dossier ftp 36](#_bookmark118)

[Figure 65 Envoi d'un fichier dans le dossier files 37](#_bookmark119)

# Sources

Smashicons : <https://www.flaticon.com/free-icon/lock_1410054>

Freepik : <https://www.flaticon.com/freeicon/process_3214710?term=process&page=1&position=33>

Wikipedia : <https://fr.wikipedia.org/wiki/Fichier:Check_green_icon.svg>

[Dimitry Miroliubov](https://www.flaticon.com/authors/dimitry-miroliubov) : <https://www.flaticon.com/freeicon/txt_337956?term=txt%20file&page=1&position=8>