**République Algérienne Démocratique et Populaire**

**Ministère de l'Enseignement Supérieur et de la recherche Scientifique**

**Université SAAD DAHLEB-Blida 1**

**Faculté des Sciences**

**Département d'Informatique**

# PROJET DE FIN D'ETUDES

Pour obtenir le titre de Master en Informatique

Option : Sécurité des systèmes d'information

**Par : Ilham KOUIDER AYAD**

Sujet

## Implémentation et teste de certains algorithmes de chiffrement et d'intégration

Soutenu devant le jury composé de :

Mr. Ben Yahia Mohammed, Département d'Informatique, Blida 1

2022/2023

# Résumé

La protection des informations sensibles est devenue une problématique importante, et la cryptographie est l'une des solutions les plus utilisées depuis longtemps pour assurer la sécurité de l'information. Elle utilise des fonctions de chiffrement pour transformer un texte clair en un texte incompréhensible, des fonctions de déchiffrement pour le reconvertir en texte clair, et des fonctions de hachage pour générer des empreintes uniques à partir des données.

Dans ce projet, nous avons implémenté plusieurs algorithmes cryptographiques pour assurer la confidentialité et l'intégrité des informations. Nous avons utilisé un algorithme cryptographique classique et simple, ainsi que l'algorithme de cryptographie asymétrique RSA, l'algorithme d'échange de clé secrète Diffie-Hellman, et l'algorithme de hachage SHA-1.



**Mots clés :** Information, sécurité de l'information, cryptographie, chiffrement, déchiffrement, fonction de hachage, algorithme classique, algorithme RSA, algorithme Diffie-Hellman, algorithme SHA-1.



# ABSTRACT

Information is a set of data that can be collected, stored, processed or disseminated. Currently, the protection of sensitive information has become a problem. Several solutions are proposed to ensure the security of information and cryptography is one of the most usable for a long time using its encryption functions to convert clear text into incomprehensible text, and decryption functions for returning to clear text, and a hash function to convert a clear text into a fixed-size fingerprint with the impossibility of returning to the initial text.

In this project we have implemented certain cryptographic algorithms to ensure the confidentiality and integrity of information: a classic and simple cryptographic algorithm, the RSA asymmetric cryptography algorithm, the Diffie-Hellman secret key exchange algorithm, and also the SHA1 hashing algorithm.



**Key words:** Information, information security, cryptography, encryption, decryption, hash function, classical algorithm, RSA algorithm, Diffie-Hellman algorithm, SHA1 algorithm.

*Je dédie ce présent travail à*

*Ma mère, qui depuis ma naissance, n'a cessé de me bercer avec des bons conseils et sans doute celle qui me réconforte et m'encourage dans le moment lesplus difficiles de ma scolarité.*

*Mon père, celui qui ne se lassait jamais pour aménager tous ses efforts pour que je sois quelqu'un dans la vie.*

*Quoi que je fasse, je ne pourrais jamais vous récompenser pour les grands sacrifices que vous avez faits et continuez de faire pour moi.*
*Aucune dédicace ne saurait exprimer mes grandes admirations, mes considérations et mes sincères affections pour vous.*

*Mes frères, Mohamed et sa femme Iman, Abderrahmane.*

*Ma sœur Sondeuse.*

*Et Islam.*

*Mes camarades de la promotion*

*Tous ceux qui de prés en prés n'ont aidé à réaliser ce mémoire*

KOUIDER AYAD Ilham

# Table des matières

## *CHAPITRE 3 IMPLEMENTATION SOUS PYCHARM*



## *CHAPITRE 4 TESTES ET RESULTATS*

# LISTE DES FIGURES

# INTRODUCTION GENERALE

## 1. contexte :

Avec l'avènement du Big Data, où les utilisateurs peuvent envoyer et recevoir des quantités massives de données, il devient essentiel d'assurer la sécurité de ces données, en particulier lorsqu'elles contiennent des informations sensibles. La cryptographie joue un rôle crucial dans la garantie de la sécurité des données.

Depuis longtemps, la cryptographie est utilisée pour dissimuler des messages à l'égard de tiers indésirables. Aujourd'hui, avec la transmission des données via Internet et leur circulation à travers diverses infrastructures, il est primordial que la cryptographie assure la fiabilité et la confidentialité des informations échangées. La cryptographie vise à préserver la confidentialité des données, tout en garantissant leur intégrité et leur authenticité.

En utilisant des techniques de chiffrement, la cryptographie permet de transformer les données en un format illisible pour les personnes non autorisées. Cela permet de préserver la confidentialité des informations et de protéger leur contenu contre d'éventuelles interceptions ou intrusions malveillantes.

De plus, la cryptographie assure également l'intégrité des données en utilisant des fonctions de hachage et de signature numérique. Les fonctions de hachage permettent de générer une empreinte unique pour chaque donnée, facilitant ainsi la détection de toute altération ou modification non autorisée. Les signatures numériques garantissent l'authenticité des données en permettant de vérifier l'identité de l'émetteur et l'intégrité du contenu.

En résumé, la cryptographie est un outil essentiel pour assurer la sécurité des données dans le contexte du Big Data. Elle permet de préserver la confidentialité, l'intégrité et l'authenticité des informations échangées, garantissant ainsi la confiance et la fiabilité des systèmes et des communications.

## 2. Problématique :

D’après la fonction de confidentialité et la fonction d’intégrité nous déduirons le problème posé à la cryptographie. Il se résout par la notion de chiffrement. Donc comment utilise la cryptographie pour assurer la sécurité d’information.

## 3. L’objectif du projet :

L’objectif principal du sujet de mémoire proposé est d’implémenter des algorithmes de cryptographie symétriques et asymétriques, ainsi que des algorithmes de hachage pour garantir que les données qui ont été envoyés, reçues ou stockées sont complets et n’ont pas modifiées et confirmées.

# 4. Organisation du mémoire :

**Le mémoire et structuré en 4 chapitres :**

Dans le premier chapitre, nous allons parler des concepts de base : l'information, la sécurité de l'information et ses principes. Ensuite, nous aborderons l'utilisation de la cryptographie pour assurer la sécurité. Enfin, nous discuterons des types d'algorithmes cryptographiques (classiques, symétriques et asymétriques), ainsi que de la fonction de hachage.

Dans le deuxième chapitre, nous allons présenter la problématique du projet et expliquer comment assurer la sécurité de l'information en utilisant certains algorithmes cryptographiques. Nous aborderons le fonctionnement de certains algorithmes, tels que RSA pour la cryptographie asymétrique, Diffie-Hellman pour l'échange de clé secret dans le contexte de la cryptographie symétrique, ainsi que SHA-1 pour vérifier l'intégrité des données.

Dans le troisième chapitre, nous allons présenter le langage de programmation utilisé (Python) ainsi que l'IDE PyCharm. Ensuite, nous aborderons l'implémentation de nos algorithmes (RSA, Diffie-Hellman, SHA-1).

Dans le quatrième chapitre, nous allons tester les algorithmes implémentés en utilisant l'affichage du terminal de Kali Linux. Nous stockerons les résultats dans des fichiers texte et analyserons les paquets à l'aide de Wireshark.

# Chapitre 1

Concepts de base

## 1.1 Introduction

Depuis des temps immémoriaux, l'homme a développé différents langages pour faciliter la communication.

La communication est un processus qui implique l'échange de données et de messages entre deux ou plusieurs utilisateurs, qui alternent entre les rôles d'expéditeur et de destinataire. Les données qui sont préparées pour composer un message sont appelées des informations. Cependant, pour garantir la sécurité de ces informations, des techniques de cryptographie peuvent être appliquées.

Dans ce chapitre, nous allons présenter les concepts de base liés à la sécurité de l'information. Nous définirons ce qu'est l'information et comment elle peut être sécurisée à l'aide de la cryptographie, en mettant l'accent sur ses trois types : la cryptographie classique ou simple, la cryptographie symétrique et asymétrique. Enfin, nous aborderons la fonction de hachage et ses différents algorithmes.

## 1.2 L'information

### 1.2.1 Etymologie

Du latin informare, façonner, former.

L'étymologie latine du terme information (du verbe informare : action de former, de façonner), ils mettent en valeur ses différents usages. [1]

### 1.2.2 Définition

En informatique et en télécommunication, l'information est un élément de connaissance (voix, données, image) susceptible d'être conservé, traité ou transmis à l'aide d'un support et d'un mode de codification, elle constituée de deux éléments :

- des données.
- un sens qui dépend de chaque individu. [1]

## 1.3 Sécurité de l'information

Le concept de sécurité de l'information c'est-à-dire doit assurer :

**La disponibilité de l'information :** doit être accessible aux personnes autorisées quand elle en a besoin.

**La confidentialité d'informations :** les informations n'appartiennent pas à tout le monde, seuls peuvent y accéder ceux qui en ont le droit.

**L'intégrité :** les informations (fichiers, messages…) ne peuvent être modifiés que par les personnes autorisées (administrateurs, propriétaires…). [1]

## 1.4 La cryptographie

### 1.4.1 Définition

L'origine de la cryptographier mot réside dans la Grèce antique. Le mot « ***crypt*** » signifie « ***caché*** » et le mot « ***graphie***r » signifie « ***écriture*** ». En informatique, la cryptographie est un ensemble des techniques pour une communication sécurisée. Elle aide à l'écriture d'un message chiffré à partir d'un message clair. En suit, écrire le message clair à partir de message chiffré.

Le chiffrement (ou cryptage) signifie la transformation d'un message en texte clair en un message chiffré par l'utilisation des opérations mathématiques indépendantes de contenu, et le but de cryptographie est le résultat du déchiffrement, c'est-à-dire, le passage de message chiffré au message en clair avec une clé privée ou publique, ou décryptage, sans connu la clé de déchiffrement par un attaquant.

Le cryptage est l'étape de chiffrement et le décryptage est l'étape de déchiffrement, mais entre ces deux étapes il y a une étape appelé la cryptanalyse qui est une technique consiste à déduire le message en clair d'un message chiffré sans posséder la clé de chiffrement. [2][3][4]

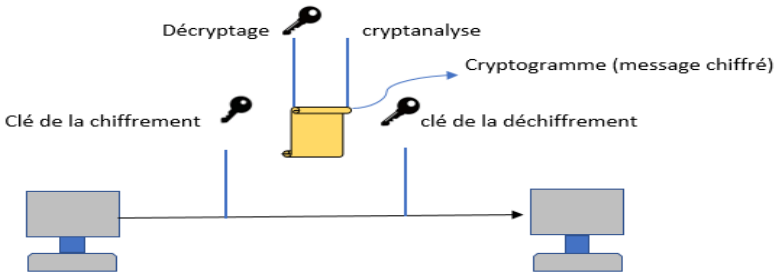


**Figure 1.1 :** cryptage, décryptage et cryptanalyse.

La figure 1.1 représente le cryptage, décryptage et cryptanalyse.

### 1.4.2 Les fonctions de la cryptographie

La cryptographie est utilisée depuis longtemps pour dissimuler des messages aux yeux de certains utilisateurs. Cette utilisation aujourd'hui avec les communications via internet circulent dans des infrastructures doit garantir la fiabilité et la confidentialité. La cryptographie sert à préserver la confidentialité des données et aussi garantir leur intégrité et leur authenticité. [4]

- **La fiabilité de l'information** est le degré de confiance que l'on peut accorder à l'information. Elle dépend de l'identification claire de la source, l'exactitude des données, des faits, la "fraîcheur' de l'information.
- **La confidentialité des données** est la protection des données stockées et des communications contre l'accès et l'interception et la lecture par les personnes qui non autorisées**.**
- **L'intégrité des données** signifie que les données qui ont été envoyés, reçues ou stockées sont complets et n'ont pas modifiées et confirmées.
- **L'authenticité** est la propriété qui permet de vérifier que l'identité de la source de données est bien prétendue.

### 1.4.3 Types de la cryptographie

D'après la fonction de confidentialité nous déduirons le premier problème posé à la cryptographie. Il se résout par la notion de chiffrement. Il existe trois types d'algorithmes cryptographiques : algorithmes classiques ou simples, algorithmes symétriques (à clef secrète), et algorithmes asymétriques (à clef publique). [5]

#### 1.4.3.a La cryptographie classique

Un système cryptographique permet de transformer un message intelligible ou message en clair M en un message chiffré incompréhensible ou cryptogramme) $c = E_{Kc}(m)$. Ce procédé est désigné sous le terme de chiffrement ou " cryptage " la procédure de chiffrement utilise un algorithme et une clé de chiffrement KC. Le message chiffré peut être mis sur un support non sécurisé. Le récepteur procédera au déchiffrement du message afin de retrouver le message en clair en appliquant la transformation inverse avec une clé de déchiffrement $\mathrm{k_d}: \mathrm{M} = \mathrm{D_{k_d}}[\mathrm{E_{kC}}(\mathrm{m})]$.

Les opérations de chiffrement reposent en général sur l'utilisation :

-D'un algorithme qui représente le modèle mathématique choisi.

-D'une clé représentant le paramètre de mise en œuvre. [6]


**Figure1.2 :** La cryptographie classique.

La figure1.2 représente le processus de la cryptographie classique simple.

### 1.4.3.b La cryptographie symétrique :

La cryptographie symétrique qui utilise une clé secrète pour le cryptage et le décryptage de l'information est l'une des plus anciennes techniques cryptographiques connues de l'homme et qui même aujourd'hui offre un haut niveau de sécurité pour faire face à différentes situations d'utilisation. [7][8]

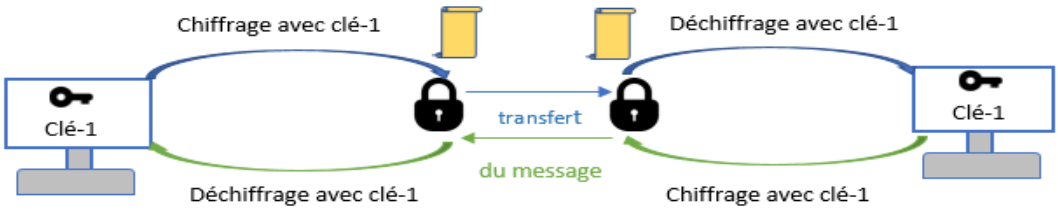


**Figure1.3 :** Le chiffrement symétrique.

La figure1.3 représente le processus de la cryptographie symétrique.

Le schéma de chiffrement symétrique repose sur une clé unique (privé) partagée entre deux utilisateurs ou plus (exemple : clé-1). La même clé est utilisée pour chiffrer et déchiffrer le texte brut (qui représente le message ou la partie de données en cours de codage). Le processus de chiffrement consiste à passer le texte brut (entrée) à travers un algorithme de cryptage, et après le transfert de du message génère un message codé (sortie).

- **Avantages :** - L'exécution de chiffrement ou déchiffrement est très rapide.
  - La clé est prédéfinie donc utilise peu de ressources systèmes.
- **Le protocole de chiffrement symétrique le plus utilisé :** est l'AES (Advanced Encryption Standard, norme de chiffrement avancé). Il développé pour mettre à jour l'algorithme DES d'origine. Parmi les applications qui utilisent l'algorithme AES les applications de messagerie comme Signal ou WhatsApp, ainsi que le programme d'archivage de fichiers Win Zip. [9]
  - **Inconvénients :** - Le chiffrement symétrique n'assure pas toutes les fonctions de cryptographie.
    - Chaque clé symétrique correspond à un échange entre 2 personnes, Pour communiquer avec d'autres personnes il faut gérer une autre clé symétrique.

- D'après l'inconvénient président il faut gérer un grand nombre de clé selon le nombre de personnes avec qui en communique.

- La garantie de la confidentialité de cette clé est très importante. [10]

### 1.4.3.c La cryptographie asymétrique :

La cryptographie asymétrique est un algorithme de déchiffrement présenté par « Diffie » et « Hellman » en 1976. Ils proposent deux clés : une clé publique peut être connue pour tous utilisée pour le chiffrement, et une clé privée pour le déchiffrement. [11] [12]

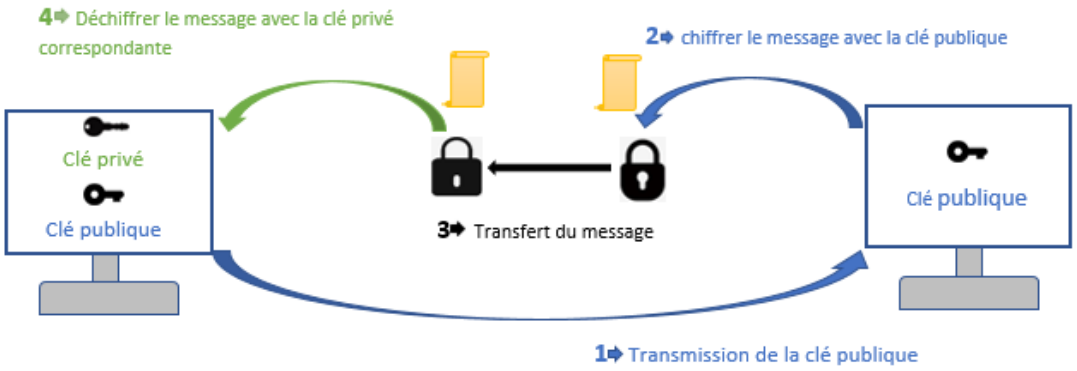


**Figure 1.4 :** Le chiffrement asymétrique.

La figure 1.4 représente le processus de la cryptographie asymétrique.

Le schéma de chiffrement asymétrique consiste à utiliser deux clés : clé de chiffrement et déchiffrement. La clé de chiffrement nommée clé publique car elle sera librement communiquée. La clé de déchiffrement nommée clé privée car elle doit connu par l'utilisateur seulement, et doit être communiquée sous aucun prétexte. [5] [13]

- **Avantage :**

Les chiffrements asymétriques utilisent la même clé publique pour communiquer avec tous les personnes, ils créent aussi moins de problèmes de gestion de clés que les chiffrements symétriques. Le nombre de clés nécessaires pour que n entités communiquent en toute sécurité entre elle est seulement 2n clés. Dans un système basé sur des chiffrements symétriques, il faudrait n (n – 1) /2 clés secrètes. [14]

## 1.5 La fonction de hachage

### 1.5.1 Définition

Une fonction de hachage est une méthode qui prend en entrée un message de taille quelconque, et applique une série de transformation et réduit ces données. On obtient à la sortie une chaine de caractères hexadécimaux, le condensé, qui résume en quelque sorte le fichier. Cette sortie a une taille fixe qui varie selon des algorithmes. Chaque fichier a son propre empreinte. Il est très difficile de retrouver ou générer un texte à partir de l'empreinte (on parle alors de fonction à sens unique). [15]

### 1.5.2 Les algorithmes de hachage

Les algorithmes de hachage, également connus sous le nom de fonctions de hachage cryptographiques, sont couramment utilisés dans le domaine de la sécurité des données pour garantir l'intégrité et la confidentialité des données. Il s'agit de fonctions mathématiques qui prennent n'importe quelle entrée (message ou données) et produisent un résultat de taille fixe appelé « hachage » ou « condensé ». La fonction de hachage doit être déterministe, c'est-à-dire que la même entrée produit toujours la même sortie, et elle doit être unidirectionnelle, c'est-à-dire qu'il est impossible d'inverser le processus et d'obtenir l'entrée originale à partir de la sortie du hachage.

Il existe deux types des algorithmes de hachage :

#### 1.5.2.a Algorithme de hachage sécurisé SHA :

L'algorithme SHA a été conçu par NSA, standardisé par NIST et publié comme FIPS PUB 180 en 1993. Une version révisée a été délivrée en 1995 comme FIPS PUB 180-1. La version actuelle est FIPS PUB 180-2 avec une notice de changement pour SHA-224 en février 2004.

La liste des algorithmes SHA est : SHA-1, SHA-256, SHA-384 et SHA-512. [16]

#### 1.5.2.b L'algorithme MD5 (Message Digest version 5) :

En 1991, Ronald Rivest améliore l'architecture de MD4 et crée MD5 (Message Digest 5). C'est une fonction de hachage cryptographique qui permet d'obtenir pour chaque message une chaine de 32 caractères (soit 128 bits) hexadécimaux avec une probabilité très forte que, pour deux messages différents, leurs empreintes soient différentes. Ce quelle que soit la taille de l'information en entrée (de 0 octets à plusieurs gigas). Cette transformation est donc irréversible (Dans le sens où on ne peut pas trouver l'information en entrée à partir d'une somme MD5). [16]

## 1.6 Conclusion

Dans ce chapitre, nous avons abordé les notions fondamentales de l'information qui nécessite une sécurisation, ainsi que la cryptographie et ses trois types : la cryptographie classique, symétrique et asymétrique, ainsi que la fonction de hachage. Ces méthodes sont largement utilisées pour assurer la sécurité de l'information.

Dans le chapitre suivant, nous allons présenter en détail les algorithmes que nous allons implémenter pour mettre en pratique ces concepts.

# Chapitre 2

Problématique

## 2.1 Introduction

Depuis de nombreux siècles, les êtres humains ont cherché à protéger leurs informations sensibles lors des communications, notamment en période de guerre. La cryptographie est l'une des techniques développées dans ce but. Cependant, avec l'avènement d'Internet et l'évolution des technologies, la cryptographie classique n'est plus suffisante pour garantir la sécurité des données échangées dans un environnement ouvert et potentiellement vulnérable.

Face à ce défi, les chercheurs en mathématiques et en informatique ont développé de nouveaux algorithmes cryptographiques modernes, tels que RSA, Diffie-Hellman et SHA1. Ces algorithmes ont été conçus pour répondre aux exigences de sécurité des communications numériques et pour protéger les informations contre les menaces potentielles.

Dans ce chapitre, nous allons présenter en détail le crypto-système RSA, son fonctionnement et son utilisation dans le domaine de la cryptographie asymétrique. Nous aborderons également l'algorithme de Diffie-Hellman, qui permet l'échange sécurisé de clés entre deux parties. Enfin, nous étudierons l'algorithme de hachage SHA1, qui est utilisé pour assurer l'intégrité des informations en générant une empreinte unique.

En complément, nous définirons également un algorithme de cryptographie simple pour illustrer les principes de base de la cryptographie. Ces connaissances nous permettront d'approfondir notre compréhension des algorithmes cryptographiques modernes et de leur application dans la sécurisation des données.

## 2.2 Le crypto système RSA

Le système de cryptage RSA a été inventé en 1977 (abréviation des noms de ses trois initiales : Ron **R**ivest, Adi **S**hamir et Len **A**dleman). Ces trois auteurs avaient décidé d'établir un nouveau système de codage révolutionnaire, appelé « système à clef publique » que W. Diffie et M. Hellman venaient d'inventer, était une impossibilité logique. Mais, ils découvrirent un nouveau système à clef publique qui supplanta vite celui de W. Diffie et M. Hellman.

Le système RSA est aujourd'hui un système utilisé dans une multitude d'applications, et appliqué par plusieurs pays et entreprises pour protégé ses données. [17] [18]

### 2.2.1 Génération des clés

Les étapes suivantes appliquées pour l'implémentation de d'l'algorithme RSA :

1) Calcul le modulo n qui est le produit de deux nombres premiers p et q : **n** = **p*****q**.
2) Calcule l'indicatrice d'Euler de n : ϕ (n) = (p − 1) * (q − 1).
3) Choisir nombre premier e tel que 3 < e < $\varphi$(n) et pgcd(e, ϕ (n)) = 1.
4) Calculer d, d'après le théorème de Bachet-Bézout, tel que : e * d ≡ 1 mod ϕ (n).
5) Nous avons clé publique : (e, n).

6) Nous avons clé privée : (d, n). Les autres valeurs doivent rester secrètes.
7) Le chiffrement : C = $M^e$ mod n, et le déchiffrement : M = $C^d$ mod n. [18]

Le chiffrement et le déchiffrement sont des calculs mathématiques, alors le résultat est des chiffres, à cause de ça doit utiliser le code ASCI pour coder un texte.

### 2.2.2 Chiffrement

Le chiffrement RSA est des calculs mathématiques, tout message doit d'abord être transforme au des nombres entiers (par exemple codé par le code ASCII). Pour le chiffrement RSA à besoin le couple (**n**, **e)** appelées la clé publique.

Le message numérique est décomposé en nombres inférieurs à **n** (taille de message(M) <n), pour chaque nombre **M**, le message numérique chiffré **C** est $c \equiv M^e (\mathrm{mod}\, n)$. [18]

### 2.2.3 Déchiffrement

Le déchiffrement nécessite de connaitre la clé privée qui le couple (d, n). Pour tout message numérique chiffré **C**, le message numérique clair **M** se calcule modulo n :

$M \equiv C^d (\mathrm{mod}\, n)$. [18]

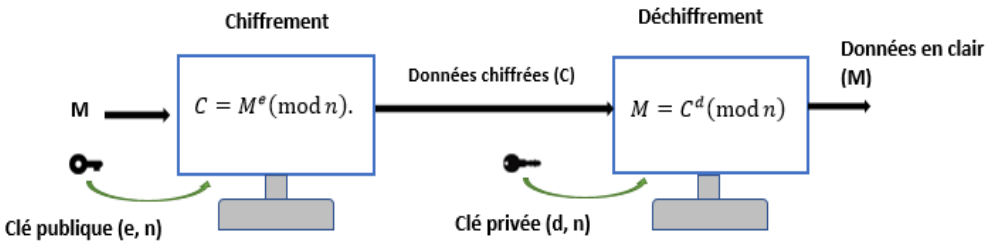


**Figure 2.1 :** Principe de fonctionnement de crypto système RSA.

La figure 2.1 représente le principe de fonctionnement de crypto système RSA.

### 2.2.4 La Sécurité du RSA

La sécurité de l'algorithme RSA est garanti, car pour casser le RSA il doit découvrir la clé privée, et il faut factoriser le nombre n. Aussi, la factorisation est un problème difficile, c'est-à-dire qu'il n'existe pas d'algorithme rapide (de complexité polynomiale) pour résoudre cette question. Mais, il se peut que les aidées soient fausses. Si c'est effectivement le cas, alors RSA n'est pas un algorithme de cryptographie sûr. [20] [21]

## 2.3 L'algorithme d'échange de clé Diffie-Hellman :

### 2.3.1 Principe de l'échange de clé de Diffie-Hellman :

L'échange de clé de Diffie-Hellman a été développé par ces deux auteurs en 1976 et publié dans l'article : W. Diffie and M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (1976), 644-654. Nous rappelons ici que l'échange d'une clé secrète est fondamental en cryptographie. En effet tout chiffrement d'une grande quantité de données ne peut se faire qu'avec du chiffrement à clé secrète, surtout si cet échange a lieu en temps réel, en raison de la lenteur relative des chiffrements à clé publique. [22]

### 2.3.2 exemple d'échange Diffie-Hellman entre deux nœuds :

Voici comment se passe l'échange Diffie-Hellman. Les calculs indiqués sont faits dans le groupe cyclique fini qui possède g comme générateur.

1. Le nœud M1 tire au hasard un entier a tel que $1 < a < P - 1$ et le garde secret.
2. Le noeud M1 envoie à M2 A=g^a mod p
3. Le noeud M2 choisit un nombre b tel que $1 < b < P - 1$ et le garde secret.
4. Le noeud M2 envoie à M1 B= g^b mod p
5. Le noeud M1 a reçu B et calcul B^a mod p (c'est-à-dire en passant par (g^b) a mod p Mais il ne connaît pas B) : S = B^a mod P
6. Le noeud M2 a reçu A et calcul A^b mod p (c'est-à-dire en passant par, (g^a) b mod p, mais il ne connaît pas A) : S= A^b mod P

M1 et M2 obtiennent à la fin de leurs calculs respectifs le même nombre qui n'a jamais été exposé à la vue des indiscrets : c'est la clé S. La figure II.2 présente le processus d'échange de clé Diffie Hellman. [22]

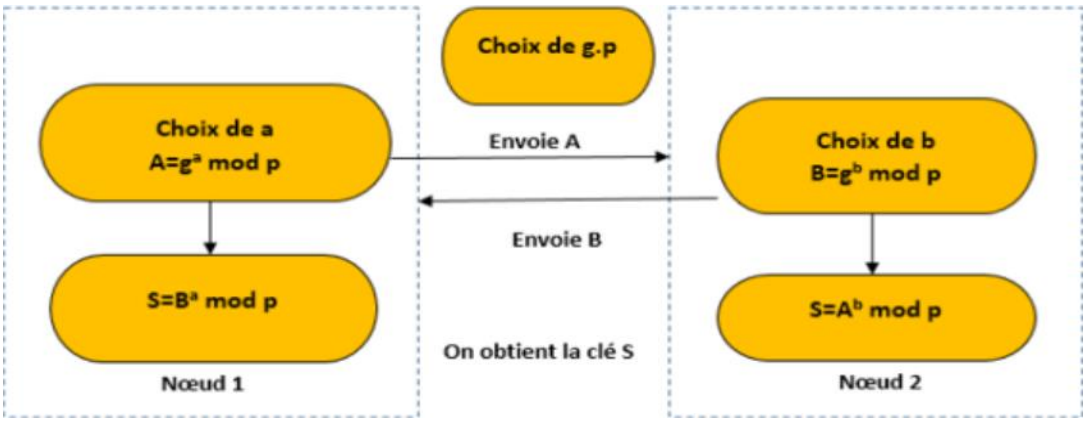


**Figure 2.2 :** Echange de clés Diffie-Hellman. [21]

La figure 2.2 représente le processus d'échange de clé sécrète de Diffie-Hellman.

## 2.4 L'algorithme de hachage SHA1 :

### 2.4.1Fonctionnement de la fonction SHA1 :

Le SHA-1 est le successeur du SHA-0 qui a été rapidement mis de côté par le NIST (national institute standards and technology) pour des raisons de sécurité insuffisante. Le SHA-0 était légitimement soupçonné de contenir des failles qui permettraient d'aboutir rapidement à des collisions (deux documents différents qui génèrent le même condensat). Face à la controverse soulevée par le fonctionnement du SHA-0 et certains constats que l'on attribue à la NSA (National Security Agency), le SHA-0 s'est vu modifié peu après sa sortie (1993) et complexifié pour devenir le SHA-1 (1995). Une collision complète sur le SHA-0 a été découverte par Antoine Joux et al. En août 2004 et laisse penser que le SHA-1 pourrait lui aussi subir une attaque.

La modification de message et la partie non linéaire aboutirent en fait à la première attaque théorique contre SHA-1. Les méthodes sont identiques à celles utilisées pour SHA-0, excepté la recherche de vecteurs de perturbation. En effet, la rotation dans l'expansion de message change complétement la situation et il a fallu créer une nouvelle heuristique pour trouver de bons candidats. [23]

### 2.4.2Les étapes de l'algorithme :

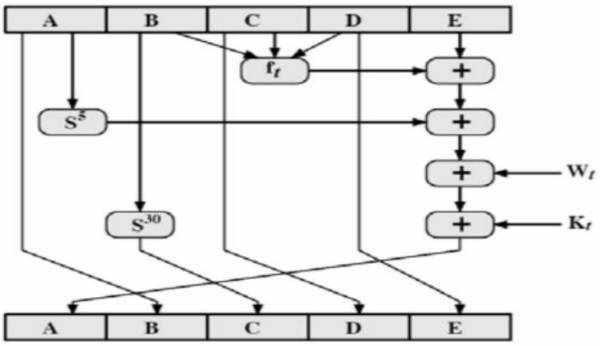


**Figure 2.3 :** Vue générale de SHA-1. [24]

La figure 2.3 représente le processus général de SHA-1.

Étape 1 : Ajout des bits d'alignement
Le message est aligné (étendue) de telle sorte que sa longueur (en bits) à satisfaire cette relation : b mod 512=448 ; à savoir le message est prolongé de manière à atteindre 64 bits plus petit qu'un multiple entier de 512. Cette expansion est toujours, indifféremment si la longueur du message satisfait déjà la relation ci-dessus. L'expansion est la suivante : ajouter un bit "1", puis ajouter des bits "0 du message pour couvrir la longueur appropriée. Au total, au moins un bit et au plus 512 bits seront ajoutés au message par cette manœuvre. [24]

Étape 2 : Ajout d'une longueur
Une représentation 64 bits de b (la longueur du message avant l'élargissement) le résultat est ajouté à l'étape précédente. En supposant par hypothèse que b est supérieur à 264, alors les moins significatifs 64 bits de b sera utilisé.  A ce stade, le message obtenu (après

l'alignement et l'extension b) a une longueur qui est un multiple entier de 512 bits. De façon équivalente, ce message a une longueur qui est un multiple entier de 16 mots (32 bits).
Ce soit la notation M [0], M [1], …, M[N-1] pour les mots du message, où N est un multiple de 16. [24]

Étape 3. Initialisation de la mémoire tampon MD
Cinq mots tampon (chaque de 32 bits) sont utilisés pour calculer le « message digest » : A, B, C, D, E. Ces registres sont chargés d'abord avec les valeurs suivantes (données ici dans la base 16) : [24]

A : 0 × 67452301

B : 0 × efcdab89

C : 0 × 98badcfe

D : 0 × 10325476

E : 0 × c3d2e1f0

Suivant, le calcul itératif consiste en utiliser le procédé illustré sur la figure 1.7. Nous avons 4 rondes de 20 itérations chacune, où nous manipulent ainsi les 5 registres :
(A, B, C, D, E) ← (E + $f_t(B, C, B)$ + (A<<5) + $w_i$ +), A, (B<<30), C, D)
Où, t est le numéro de l'étape, f (t, B, C, D) est une fonction primitive non-linéaire de la ronde pour l'étape t, est dérivé du bloc de message (32 bits) et est une valeur additive.
Les fonctions primitives sont utilisées comme suit : [23]
0 ≤ t ≤ 19 $f_t$ (B, C, D) = (B ∧ C ) ∨ ( ¬B ∧ D)
20 ≤ t ≤ 39 $f_t$ (B, C, D) = B ⊕ C ⊕ D
40 ≤ t ≤ 59 $f_t$ (B, C, D) = (B ∧ C ) ∨ (B ∧ C ) ∨ ( C ∧ D)
60 ≤ t ≤ 79 $f_t$ (B, C, D) = B ⊕ C ⊕ D

$w_i$ est calculé comme suit :

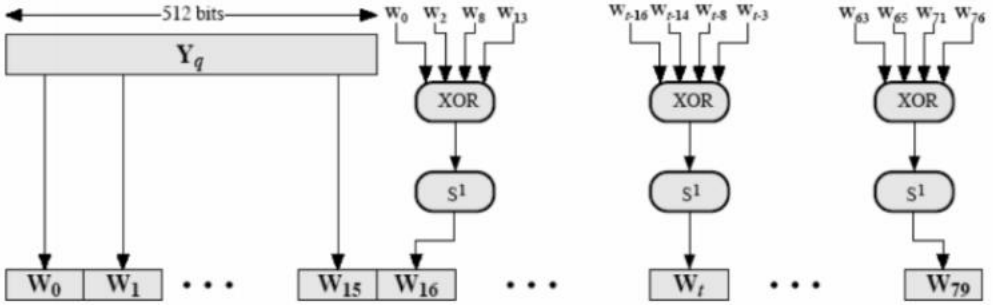


**Figure 2.4 :** Calcul de $w_i$ . [21]

La figure 2.4 représente le processus de calcul de $w_i$.

Pour 0 ≤ t ≤ 15, prend la valeur les 16 premières valeurs du bloc
Pour 16 ≤ t, $w_i = w_{i-16} \oplus w_{i-14} \oplus w_{i-8} \oplus w_{i-3}$
Enfin, la valeur additive $k_t$, prend ces valeurs comme suit : [21]
0 ≤ t ≤ 19 $k_t$ = 0x 5a 827999
20 ≤ t ≤ 39 $k_t$ = 0x 6ed9eba1

$40 \leq t \leq 59$ $k_t$ = 0x 8f 1 bbcdc
$60 \leq t \leq 79$ $k_t$ = 0xca62c1d 6

## 2.5 Algorithme de cryptographie simple

Un algorithme de cryptographie simple est une méthode de transformation d'un message en clair en un message chiffré incompréhensible, à l'aide d'une formule mathématique prédéfinie avec une clé.

### 2.5.1 L'algorithme implémenté

Dans ce projet nous allons implémente un algorithme simple de cryptographie avec deux fonctions de chiffrement et déchiffrement biens définie :

#### 2.5.1.a chiffrement :

Etape 1 : convertir le message de type chaine de caractères en code ascii.

Etape 2 : convertir le code ascii de type décimale en binaire.

Etape 3 : convertir la clé d'expéditeur qui échangée par l'algorithme Diffie_Hellman de type décimale en binaire.

Etape 4 : applique la fonction « ou exclusif » qui appelée XOR en binaire entre le message et la clé en binaire. $c_b = M_b \oplus k_b$

Etape 5 : convertir le résultat de type binaire en décimale.

Etape 6 : lire le résultat décimal comme code ascii et le convertir en chaine de caractères.

#### 1.5.1.b déchiffrement :

On a déchiffré avec la même opération mais dans le sens inverse avec les étapes suivantes :

Etape 1 : convertir le message chiffré de type chaine de caractères en code ascii.

Etape 2 : convertir le code ascii de type décimale en binaire.

Etape 3 : convertir la clé de destinataire qui échangée par l'algorithme Diffie-Hellman de type décimale en binaire.

Etape 4 : applique la fonction « ou exclusif » qui appelée XOR en binaire entre le message et la clé en binaire. $M_b = c_b \oplus k_b$

Etape 5 : convertir le résultat de type binaire en décimale.

Etape 6 : lire le résultat décimal comme code ascii et le convertir en chaine de caractères.

## 2.6 Conclusion

Dans ce chapitre, nous avons exploré différents exemples d'algorithmes de cryptographie, notamment le crypto-système RSA, l'algorithme de Diffie-Hellman et l'algorithme de hachage SHA1. Ces exemples représentent des avancées significatives dans le domaine de la cryptographie moderne et sont largement utilisés pour assurer la sécurité des communications et la protection des informations sensibles.

Dans le prochain chapitre, nous nous concentrerons sur les outils nécessaires à l'implémentation de ces algorithmes. Nous présenterons le langage de programmation et l'environnement de développement intégré (IDE) que nous utiliserons pour mettre en œuvre ces algorithmes. Cette étape est essentielle pour concrétiser les concepts théoriques et les transformer en code fonctionnel.

En combinant les connaissances acquises sur les algorithmes de cryptographie avec les compétences en programmation, nous serons en mesure de mettre en place des solutions robustes pour assurer la sécurité des données.

# Chapitre 3

Implémentation sous PyCharm

## 3.1 Introduction

Dans le domaine du développement de logiciels, il existe aujourd'hui une multitude de langages de programmation, qui sont généralement classés en quatre catégories principales : les langages pour les sites statiques, les langages pour les sites dynamiques, les langages pour les applications mobiles et les langages de programmation pour l'intelligence artificielle.

Chaque catégorie de langages possède ses propres caractéristiques et est utilisée dans des contextes spécifiques. De plus, chaque langage est généralement accompagné d'un environnement de développement intégré (IDE) qui facilite l'écriture, le test et le débogage du code.

Dans ce chapitre, nous nous concentrerons sur le langage de programmation Python, qui est largement utilisé pour une variété d'applications, allant du développement web à l'analyse de données en passant par l'intelligence artificielle. Nous présenterons également l'IDE PyCharm, qui est l'un des environnements les plus populaires pour le développement en Python.

Nous aborderons les fonctionnalités clés de Python et de PyCharm, et nous explorerons les codes que nous avons écrits pour mettre en œuvre les algorithmes choisis. Cette étape est essentielle pour transformer les concepts théoriques des algorithmes en code fonctionnel, prêt à être exécuté et testé.

En combinant les avantages du langage Python et de l'IDE PyCharm, nous serons en mesure de développer et de mettre en œuvre nos algorithmes de cryptographie de manière efficace et productive.

## 3.2 Installation de python et PyCharm

### 3.2.1 Installation de python

Le langage de programmation Python a été créé en 1989 par Guido van Rossum, aux Pays-Bas. Le nom Python vient d'un hommage à la série télévisée Monty Python's Flying Circus dont G. van Rossum est fan. La première version publique de ce langage a été publiée en 1991. La dernière version de Python est la version 3. Plus précisément, la version 3.7 a été publiée en juin 2018. La version 2 de Python est désormais obsolète et cessera d'être maintenue après le 1er janvier 2020. Dans la mesure du possible évitez de l'utiliser. [25]

### 3.2.2 Les étapes d'installation sur kali linux

Etape 1 : premièrement vérifier s'il existe par défaut python sur la machine avec la commande : python3 --version.

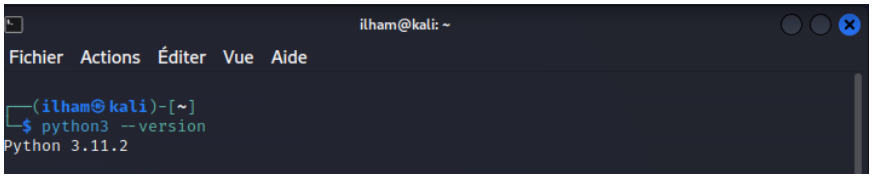


**Figure 3.1 :** vérifier la version de python.

Etape 2 : s'il n'existe pas par défaut python sur votre machine ou vous voulez installer une autre version premièrement exécuter la commande : sudo apt update, pour sauvegarder les modifications.

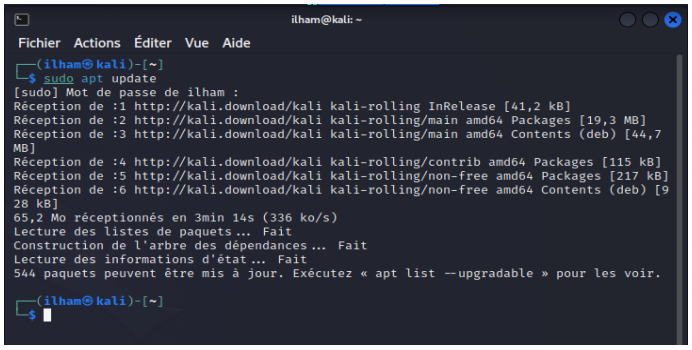


**Figure 3.2 :** sauvegarder les modifications.

Etape 3 : Après exécuter la commande : sudo apt install python3, avec cette commande la machine va installer par défaut la dernière version 3.11.

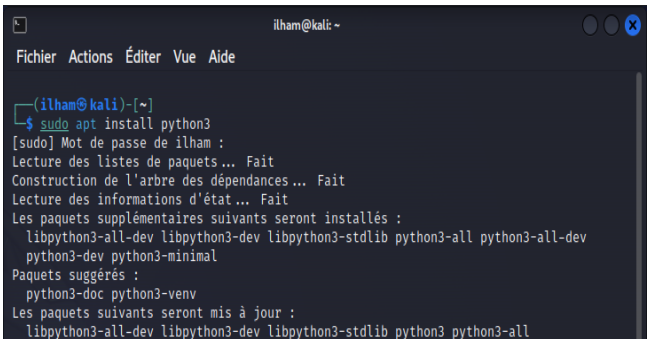


**Figure 3.3 :** installer la version de python.

### 3.2.3 Installation de PyCharm

PyCharm est un environnement de développement intégré utilisé en programmation spécialement pour le langage Python. Il est développé par la société tchèque JetBrains. Il fournit une analyse de code, un débogueur graphique, un testeur d'unité intégré.

PyCharm est multiplateforme, avec les versions Windows, MacOs et Linux [26].

### 3.2.4 Les étapes d'installation sur kali linux

Etape 1 : à partir de site https://www.jetbrains.com/pycharm/download/#section=windows télécharger la version Community For pure Python development in linux.

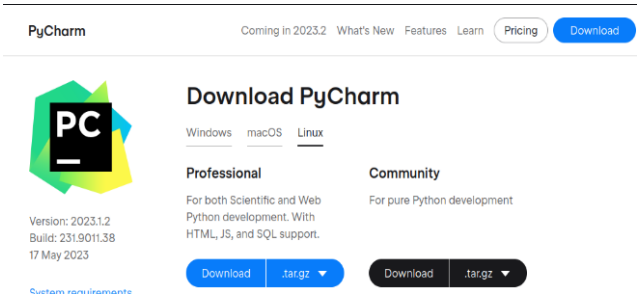


**Figure 3.4 :** téléchargement de PyCharm Community.

Etape 2 : et après changer le répertoire de fichier téléchargé (nouveau répertoire est : le fichier opt), et extraire le fichier pycharm-community-2023.1.2.tar.gz.

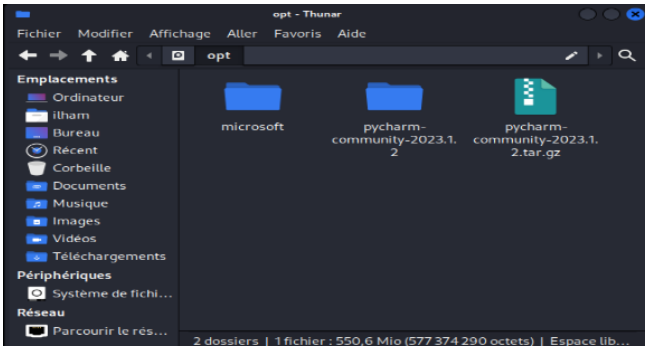


**Figure 3.5 :** extraire de fichier tar.gz dans opt.

Etape 3 : l'étape suivante aller au kali terminal et ouvrer le fichier opt, ensuite ouvrer le fichier pycharm-community-2023.1.2, ensuite le fichier bin, ensuite exécuter la commande ./pycharm.sh.


**Figure 3.6 :** accéder au fichier bin.

Etape 4 : pour démarrer l'installation, veuillez accepter les conditions et après cliquer sur « continue » sur la fenêtre affichée.

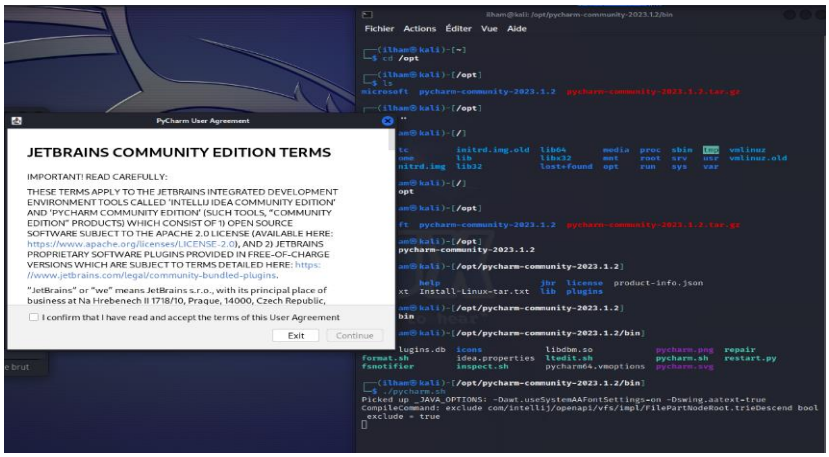


**Figure 3.7 :** lancer l'installation de PyCharm.

## 3.3 Implémentation

### 3.3.1 Création de premier projet sur PyCharm

Quand l'installation de PyCharm est terminée, nous avons créé le premier projet :

- Clique sur « New Project ».
- Ecrire le nom de projet « encryption_algorithmes » dans « Localisation ».
- Clique sur le bouton Create.

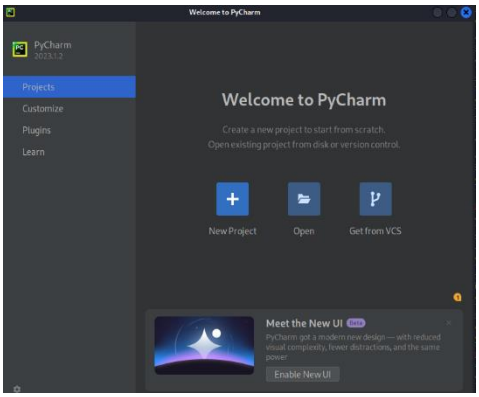


**Figure 3.8 :** clique sur New Project.

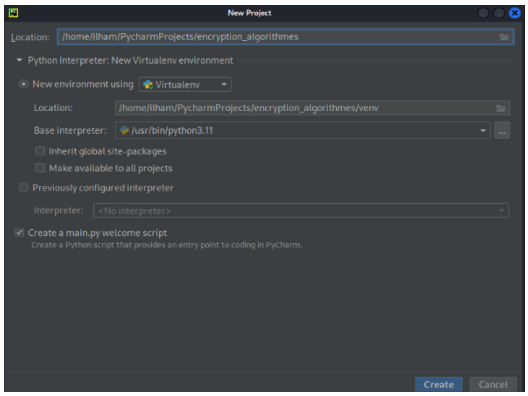


**Figure 3.9 :** écrire le nom de projet.

### 3.3.2 Implémentation d'algorithme RSA

Etape 1 : Premièrement nous avons créé un fichier python : clic droit sur le projet « encryption_algorithmes », New, Python file, écrire le nom de fichier « rsa_encryption_fille».

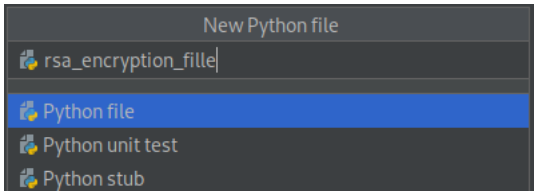


**Figure 3.10 :** écrire le nom de fichier.

Etape 2 : on les deux premières lignes de fichier import les deux modules python « random et math » pour utiliser ses fonctions prédéfinis.

```
import random
import math
```

**Figure 3.11 :** import moules.

Etape 3 : écrire la première fonction « is_prime » pour vérifier si un nombre est premier.

```
def is_prime(number):
    if number < 2:
        return False
    for k in range(2, number // 2 + 1):
        if number % k == 0:
            return False
    return True
```

**Figure 3.12 :** fonction is_prime.

Etape 4 : écrire une fonction « generate_prime » pour générer un nombre premier.

```
def generate_prime(min_value, max_value):
    prime = random.randint(min_value, max_value)
    while not is_prime(prime):
        prime = random.randint(min_value, max_value)
    return prime
```

**Figure 3.13 :** fonction generate_prime.

Etape 5 : écrire une fonction « mod_inverse » pour calculer le modinverse.

```
def mod_inverse(r, g):
    for j in range(3, g):
        # if (j * r) % g == 1:
        if (r % g) * (j % g) % g == 1:
            return j
    raise ValueError("mod_inverse does not exist")
```

**Figure 3.14 :** fonction mod_inverse.

Etape 6 : appelle fonction « generate_prime » pour générer les deux nombres p et q qui ne sont pas égaux, calcul n et z, générer la clé publique e tel que e et z sont premier entre aux, appelle fonction « mod_inverse » pour calculer la clé prévit d le modinverse de e.

```python
p, q = generate_prime(1000, 5000), generate_prime(1000, 5000)

while p == q:
    q = generate_prime(1000, 5000)

n = p * q
z = (p - 1) * (q - 1)

e = random.randint(3, z - 1)
while math.gcd(e, z) != 1:
    e = random.randint(3, z - 1)

d = mod_inverse(e, z)

print("public key: ", e)
print("private key: ", d)
print("n: ", n)
print("z: ", z)
print("p: ", p)
print("q: ", q)
```

**Figure 3.15 :** génération une clé publique et une clé prévit.

Etape 7 : lire le contenue d'un fichier choisi et chiffrer le code ascii de ce fichier avec la clé publique e, écrire le résultat dans un autre fichier.

```python
filename = input("Please enter a filename. ")
# message = " "
with open(filename, 'r') as file:
    message = file.read()
print(message)
message_encoded = [ord(ch) for ch in message]
print(message_encoded)
# (m ^ e) mod n = c
ciphertext = [pow(ch, e, n) for ch in message_encoded]
print(ciphertext)

ss = str(ciphertext)
with open('f_1', 'w') as file:
    file.write(ss)
```

**Figure 3.16 :** cryptage d'un fichier.

Etape 8 : déchiffre le message chiffré avec la clé prévit d, écrire le résultat dans un autre fichier.

```python
message_encoded = [pow(ch, d, n) for ch in ciphertext]
print(message_encoded)
message = "".join(chr(ch) for ch in message_encoded)

print(message)

with open('f_2', 'w') as file:
    file.write(message)
```

**Figure 3.17 :** décryptage d'un message chiffré.

### 3.3.3 Implémentation d'échange de clé Diffie-Hellman

Etape 1 : Premièrement nous avons créé un fichier python : clic droit sur le projet « encryption_algorithmes », New, Python file, écrire le nom de fichier « Deffie_Hellman_algorithm », import le module random, générer un nombre aléatoire p, créer un matrice n*m tel que : M [i, j] = i ^ j mod p, lire la liste des indices des lignes qui ne contient pas le chiffre 1, choisir un nombre de cette liste.

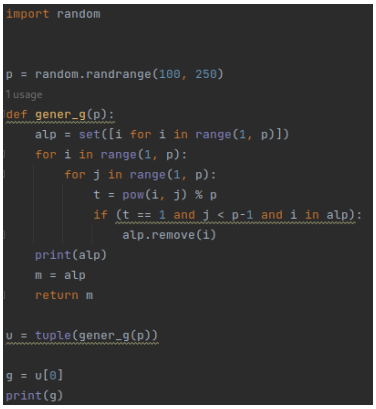


**Figure 3.18 :** choisir p et génère g.

Etape 2 : génère une clé prévit a pour l'expéditeur et une clé prévit b pour le destinataire, et calcule les clés publiques A et B avec la formule : A= g^ a mod p, B= g^ b mod p, calcule la clé échangée avec la formule : K= x^ a mod p tel que y est la clé publique de l'expéditeur et x est la clé prévit du destinataire.


**Figure 3.19 :** calcul de la clé échangée.

### 3.3.4 Implémentation d'algorithme de cryptographie simple

Etape 1 : dans le même fichier président nous avons implémenter un algorithme simple pour chiffrer le contenue d'un fichier avec l'opération binaire XOR entre le texte clair et la clé échangée par Diffie-Hellman, la fonction « Encrypt ».

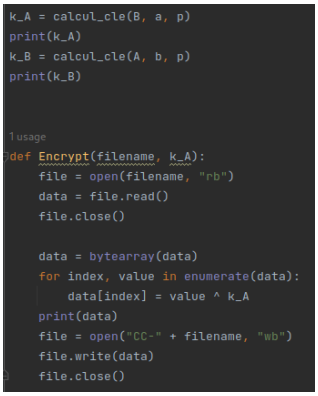


**Figure 3.20 :** chiffrage simple avec la clé échangée par Diffie-Hellman.

Etape 2 : déchiffrage simple de texte chiffré président avec l'opération binaire XOR entre le texte chiffré et la clé échangée par Diffie-Hellman, la fonction « Encrypt ».

```
def Decrypt(filename, k_B):
    file = open(filename, "rb")
    data = file.read()
    file.close()

    data = bytearray(data)
    for index, value in enumerate(data):
        data[index] = value ^ k_B
    print(data)
    file = open("DC-" + filename, "wb")
    file.write(data)
    file.close()


filename = input("Enter filename :\n")

Encrypt(filename, k_A)
Decrypt("CC-" + filename, k_B)
```

**Figure 3.21 :** déchiffrage simple avec la clé échangée par Diffie-Hellman.

### 3.3.5 Implémentation de l'algorithme SHA1

Etape 1 : Premièrement nous avons créé un fichier python : clic droit sur le projet « encryption_algorithmes », New, Python file, écrire le nom de fichier « sha1_file », écrire la première fonction « sha1 », définie les cinq constantes $H_0, H_1, H_2, H_3, H_4$, ensuite convertir le message en binaire et ajoute '1' à droite, ensuite ajoute des zéro '0' jusqu'à la taille de message en binaire mod 512 égale 448, et après convertir la taille de message en binaire et ajouter des zéro '0' à gouache jusqu'à 64 bits, enfin ajoute résultat à la fin de premier résultat.

```
def sha1(data):
    bytes = ""

    h0 = 0x67452301
    h1 = 0xEFCDAB89
    h2 = 0x98BADCFE
    h3 = 0x10325476
    h4 = 0xC3D2E1F0

    for n in range(len(data)):
        bytes+='{0:08b}'.format(ord(data[n]))
    bits = bytes+"1"
    pBits = bits
    #pad until length equals 448 mod 512
    while len(pBits)%512 != 448:
        pBits+="0"
    #append the original length
    pBits+='{0:064b}'.format(len(bits)-1)
```

**Figure 3.22 :** les calcul binaire nécessaires.

Etape 2 : écrire la fonction chunks pour découpe le résultat d'étape 1 en des parties de même taille, et la fonction rol pour calcul mots entre 16 et 80.

```
def chunks(l, n):
    return [l[i:i+n] for i in range(0, len(l), n)]

def rol(n, b):
    return ((n << b) | (n >> (32 - b))) & 0xffffffff
```

**Figure 3.23 :** les fonctions chunks et rol.

Etape 3 : pour chaque chunk écrire les mots tel que les 16 mots premier sont les même mais les autre jusqu'à 80 sont calculés par la formule $w_i = w_{i-16} \oplus w_{i-14} \oplus w_{i-8} \oplus w_{i-3}$, et mettre les valeurs constantes $H_0, H_1, H_2, H_3, H_4$ dans autres variables a, b, c, d, e.

```
for c in chunks(pBits, 512):
    words = chunks(c, 32)
    w = [0]*80
    for n in range(0, 16):
        w[n] = int(words[n], 2)
    for i in range(16, 80):
        w[i] = rol((w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16]), 1)

    a = h0
    b = h1
    c = h2
    d = h3
    e = h4
```

**Figure 3.24 :** calcul des mots.

Etape 3 : applique quatre fonctions sur les 80 mots tel que : première fonction appliquée sur les mots entre 1 et 20 : $f_t$ (B, C, D) = (B ∧ C ) ∨ ( ¬B ∧ D),la deuxième appliquée entre 21 et 40 : $f_t$ (B, C, D) = B ⊕ C ⊕ D, la troisième entre 41 et 60 :

$f_t$ (B, C, D) = (B ∧ C ) ∨ (B ∧ C ) ∨ ( C ∧ D), la quatrième entre 61 et 80 :

$f_t$ (B, C, D) = B ⊕ C ⊕ D, enfin retourne le code hache.

```
    #Main loop
    for i in range(0, 80):
        if 0 <= i <= 19:
            f = (b & c) | ((~b) & d)
            k = 0x5A827999
        elif 20 <= i <= 39:
            f = b ^ c ^ d
            k = 0x6ED9EBA1
        elif 40 <= i <= 59:
            f = (b & c) | (b & d) | (c & d)
            k = 0x8F1BBCDC
        elif 60 <= i <= 79:
            f = b ^ c ^ d
            k = 0xCA62C1D6

        temp = rol(a, 5) + f + e + k + w[i] & 0xffffffff
        e = d
        d = c
        c = rol(b, 30)
        b = a
        a = temp

    h0 = h0 + a & 0xffffffff
    h1 = h1 + b & 0xffffffff
    h2 = h2 + c & 0xffffffff
    h3 = h3 + d & 0xffffffff
    h4 = h4 + e & 0xffffffff

return '%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4)
```

**Figure 3.20 :** calcul l'empreinte de SHA1.

## 3.4. Conclusion

Dans ce chapitre, nous avons implémenté certains algorithmes de cryptographie pour assurer la confidentialité des informations par le chiffrement et l'intégrité par le hachage.

Dans le chapitre suivant, nous allons tester les algorithmes de cryptographie implémentés dans ce chapitre.

# Chapitre 4

Teste et résultats

## 4.1 Introduction

Dans le chapitre précédent, nous avons présenté les codes écrits pour implémenter l'algorithme de cryptage RSA, l'algorithme d'échange de clé Diffie-Hellman, l'algorithme de chiffrement simple, ainsi que l'algorithme de hachage SHA1.

Dans ce chapitre, nous allons tester les codes Python en les exécutant dans le terminal de Kali Linux et vérifier le contenu des fichiers manipulés. Ensuite, nous allons écrire un code Python pour mettre en œuvre une discussion cryptée avec RSA et analyser les paquets à l'aide de Wireshark.

## 4.2 Teste sur kali linux terminal

### 4.2.1 Teste d'algorithme RSA

Premièrement ouvrir le terminal de kali linux, ensuite ouvrir le répertoire encryption_algorithmes avec la commande « cd », ensuite exécute le fichier « rsa_encryption_fille » avec la commande « python3 rsa_encryption_fille.py », pour chiffre le fichier test.

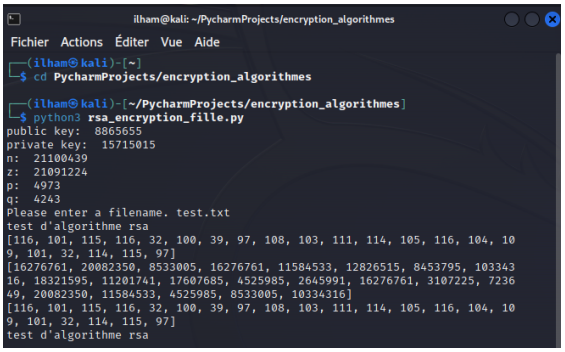


**Figure 4.1 :** teste d'algorithme RSA.

### 4.2.2 Teste d'algorithme d'échange de clé Diffie-Hellman et l'algorithme de chiffrement simple

Exécute le fichier « deffie_hellman_algorithm » avec la commande «python3 deffie_hellman_algorithm.py ». Calcul de clé échangée par Diffie-Hellman, ensuite chiffrement et déchiffrement simple avec cette clé.

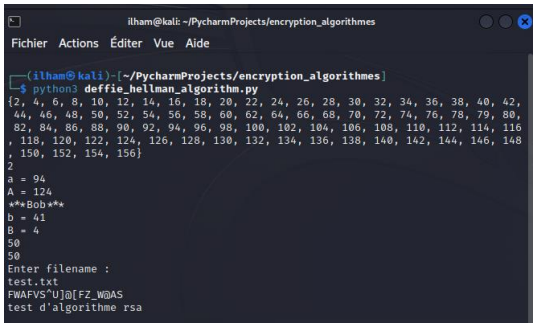


**Figure 4.2 :** teste d'algorithme Diffie-Hellman et d'algorithme de chiffrement simple.

### 4.2.3 Teste d'algorithme SHA1

Exécute le fichier « sha1_file » avec la commande « python3 sha1_file.py ». Calcul le code hache de fichier choisi, ensuite compare le code hache de fichier et le code hache de fichier déchiffré avec l'algorithme simple président.

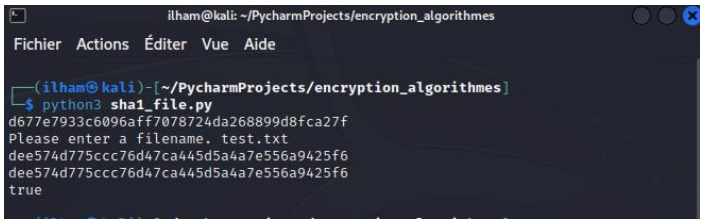


**Figure 4.3 :** teste d'intégrité à l'aide d'algorithme SHA1.

## 4.3 manipulation sur les fichiers

### 4.3.1 Teste d'algorithme RSA

Pour l'algorithme RSA on a trois fichier : fichier qui contient le texte clair, le deuxième fichier contient le texte chiffré, et le dernier contient le texte déchiffré.

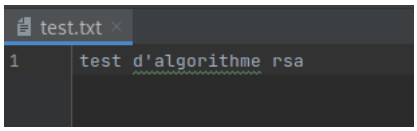


**Figure 4.4 :** fichier de texte clair.

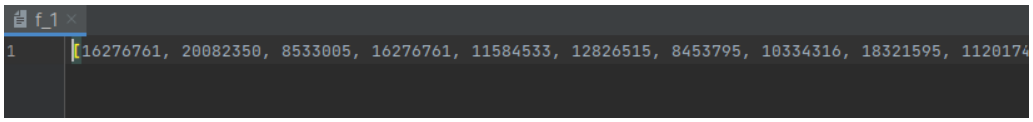


**Figure 4.5 :** fichier de texte chiffré.

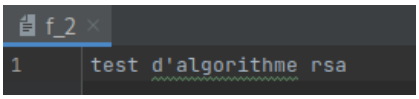


**Figure 4.6 :** fichier de texte déchiffré.

### 4.3.2 Teste d'algorithme de cryptographie simple

Pour l'algorithme de chiffrement/déchiffrement simple on a aussi trois fichier : fichier qui contient le texte clair, le deuxième fichier contient le texte chiffré avec la clé échangée par Diffie-Hellman, et le dernier contient le texte déchiffré avec la même clé.

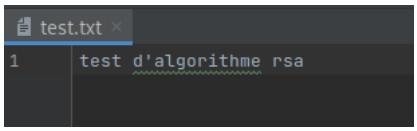


**Figure 4.7 :** fichier de texte clair.

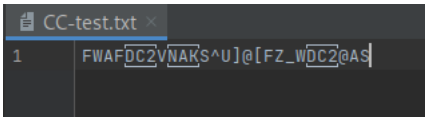


**Figure 4.8 :** fichier de texte chiffré.

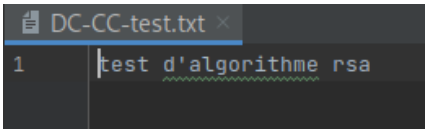


**Figure 4.9 :** fichier de texte déchiffré.

## 4.4 Analyser les paquets avec Wireshark

### 4.4.1 Wireshark

Wireshark est un outil d'analyse de protocoles réseaux. Disponible sur système d'exploitation classique (Windows 32 et 64 bits, etc.), cet analyseur capture le trafic de réseaux variés : Ethernet, TCP, Bluetooth... Wireshark stocke les données acquises dans un fichier dédié. En licence gratuite, open source.

Pour installer Wireshark premièrement nous avons ouvrir le terminal de kali linux et exécute la commande « sudo apt-get install wireshark », ensuite la commande « wireshark » pour l'ouvrir.

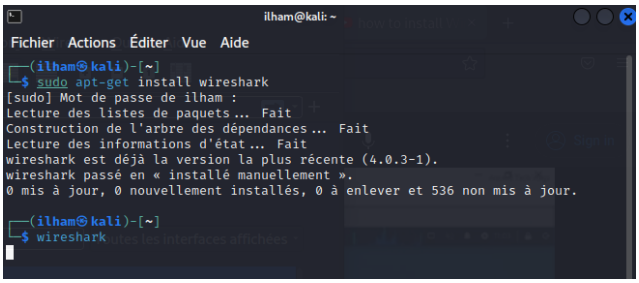


**Figure 4.10 :** installe wireshark.

### 4.4.2 Discussion simple en python

#### 4.4.2.a le code python de discussion simple

Premièrement nous avons importé les modules « socket » et « threading » pour gérer les connections entre de clients avec le protocole TCP par une adresse locale et un numéro de port.

```python
import socket
import threading



choice = input("Do you want to host (1) or to connect (2): ")
if choice == "1":
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(("127.0.0.1", 65530))
    server.listen()

    client, _ = server.accept()

elif choice == "2":
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(("127.0.0.1", 65530))


else:
    exit()
```

**Figure 4.11 :** implémente l'architecteur server client avec TCP.

Deuxièmes nous avons défini les deux fonctions de pour envoyer le message et pour recevoir le message entre deux clients.

```python
def sending_messages(c):
    while True:
        message = input("")
        c.send(message.encode())
        print("you: " + message)

1 usage
def receiving_messages(c):
    while True:
        print("Partner: " + c.recv(1024).decode())

threading.Thread(target=sending_messages, args=(client,)).start()
threading.Thread(target=receiving_messages, args=(client,)).start()
```

**Figure 4.12 :** démarre la discussion.

### 4.4.2.b analyse des paquets

Etape 1 : ouvrir wireshark avec la commande de kali linux terminal, ensuite choisi Options sur Capture.

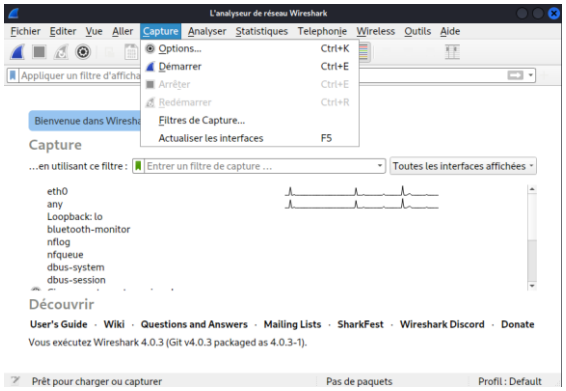


**Figure 4.13 :** choisi le trafic capturé.

Etape 2 : lorsque nous avons utilisé le protocole TCP alors nous avons choisi l'interface Loopback, ensuite clique sur démarrer.

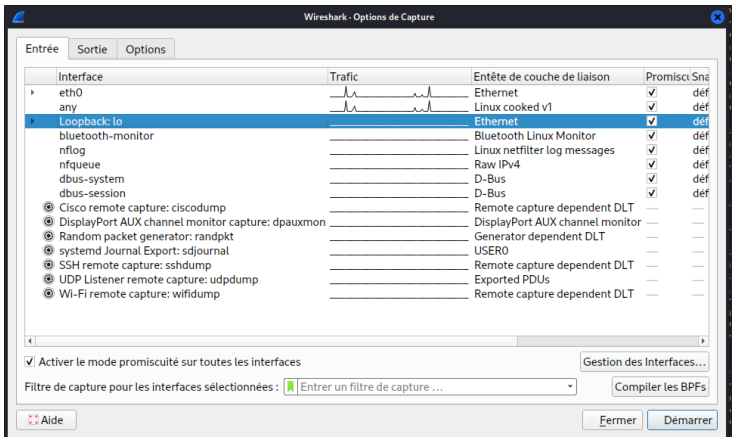


**Figure 4.14 :** lance l'analyse.

Etape 3 : lance une nouvelle instance de terminal, ouvrir le répertoire « PycharmProjects/encryption_algorithmes », exécute la commande « python3 simple_chat.py », choisi 1 pour le premier client, lance la discussion.

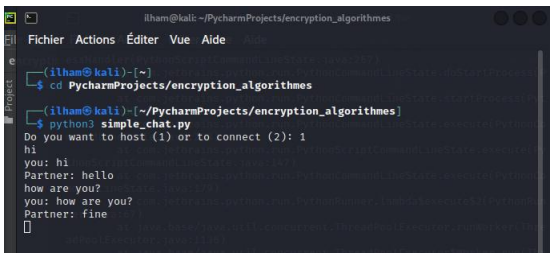


**Figure 4.15 :** lance la discussion par le premier client.

Etape 4 : lance une nouvelle instance de terminal, ouvrir le répertoire « PycharmProjects/encryption_algorithmes », exécute la commande « python3 simple_chat.py », choisi 2 pour le deuxième client, l'échange des messages entre les deux clients.

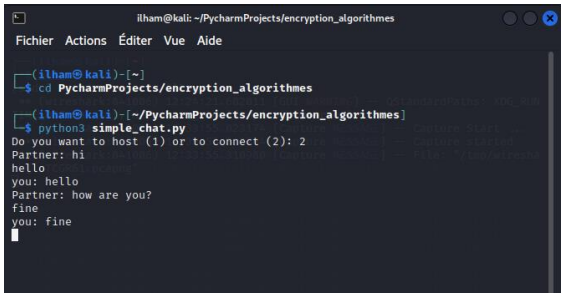


**Figure 4.16 :** échange des messages entre les deux clients.

Etape 4 : analyse des paquets, premier paquet pour le premier message de client 1 envoyé à client 2, deuxième paquet pour le premier message de client 2 envoyé à client 1.

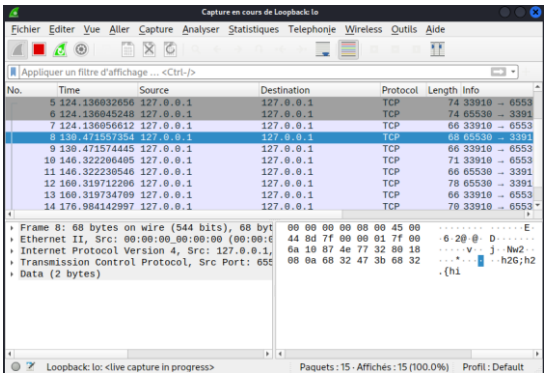


**Figure 4.17 :** premier message de client 1 « hi ».

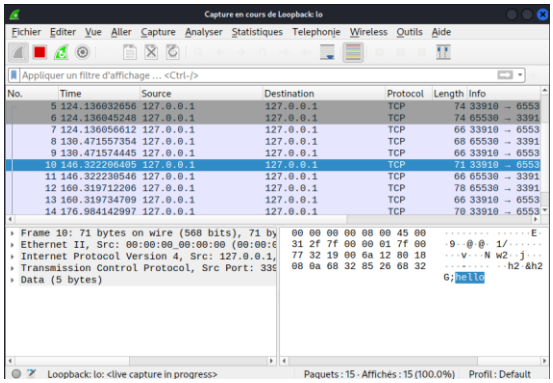


**Figure 4.18 :** premier message de client 2 « hello ».

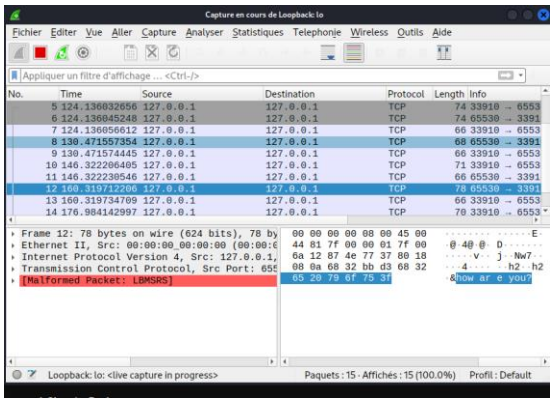


**Figure 4.19 :** deuxième message de client 1 « how are you? ».

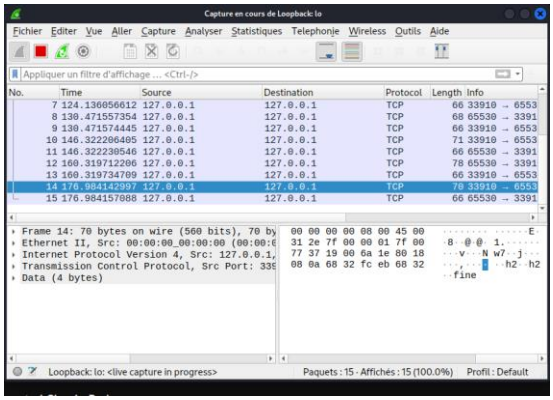


**Figure 4.20 :** deuxième message de client 2 « fine ».

### 4.4.3 Discussion cryptée

### 4.4.3.a le code python de discussion cryptée

Premièrement nous avons importé les modules « socket » et « threading » pour gérer les connections entre de clients avec le protocole TCP par une adresse locale et un numéro de port, aussi import le module « rsa » pour crypter les messages.

```python
import socket
import threading

import rsa

public_key, private_key = rsa.newkeys(1024)
public_partner =None

choice = input("Do you want to host (1) or to connect (2): ")
if choice == "1":
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(("127.0.0.1", 65530))
    server.listen()

    client, _ = server.accept()
    client.send(public_key.save_pkcs1("PEM"))
    public_partner = rsa.PublicKey.load_pkcs1(client.recv(1024))
elif choice == "2":
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(("127.0.0.1", 65530))
    client.send(public_key.save_pkcs1("PEM"))
    public_partner = rsa.PublicKey.load_pkcs1(client.recv(1024))

else:
    exit()

1 usage
```

**Figure 4.21 :** implémente l'architecteur server client avec TCP.

```python
def sending_messages(c):
    while True:
        message = input("")
        c.send(rsa.encrypt(message.encode(), public_partner))
        print("you: " + message)

1 usage
def receiving_messages(c):
    while True:
        print("Partner: " + rsa.decrypt(c.recv(1024), private_key).decode())

threading.Thread(target=sending_messages, args=(client,)).start()
threading.Thread(target=receiving_messages, args=(client,)).start()
```

**Figure 4.22 :** chiffre et déchiffre les messages à l'aide de RSA.

### 4.4.3.b analyse des paquets

Etape 1 : ouvrir wireshark avec la commande de kali linux terminal, ensuite choisi Options sur Capture, choisi l'interface Loopback, ensuite clique sur démarrer.

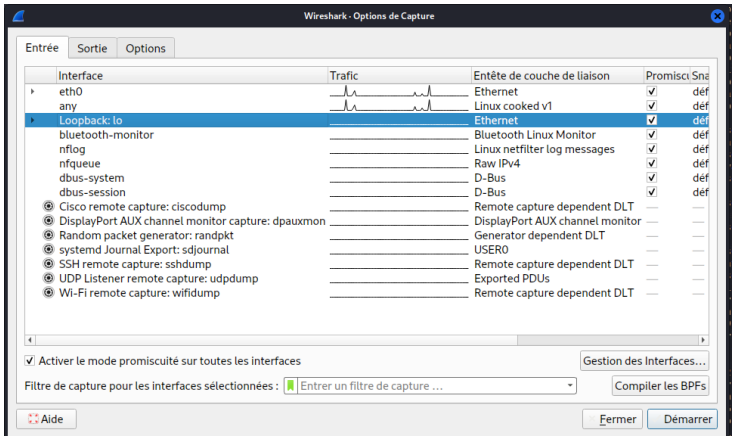


**Figure 4.23 :** choisi l'interface et démarre l'analyse.

Etape 2 : lance une nouvelle instance de terminal, ouvrir le répertoire « PycharmProjects/encryption_algorithmes », exécute la commande « python3 rsa_encryption_chat.py », choisi 1 pour le premier client, lance la discussion.

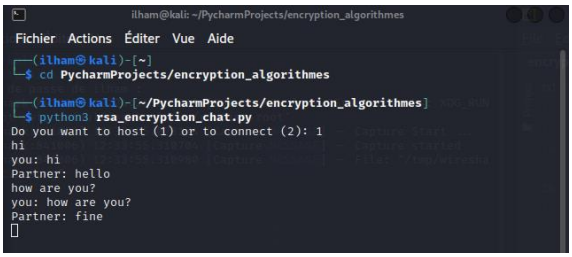


**Figure 4.24 :** l'interface de client 1.

Etape 3 : lance une nouvelle instance de terminal, ouvrir le répertoire « PycharmProjects/encryption_algorithmes », exécute la commande « python3 rsa_encryption_chat.py », choisi 1 pour le premier client, lance la discussion.

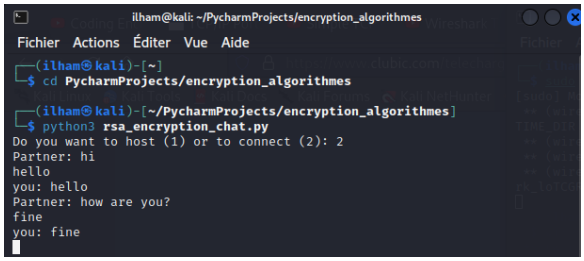


**Figure 4.25 :** l'interface de client 2.

Etape 4 : analyse des paquets, premier paquet pour le premier message chiffré de client 1 envoyé à client 2, deuxième paquet pour le premier message chiffré de client 2 envoyé à client 1.

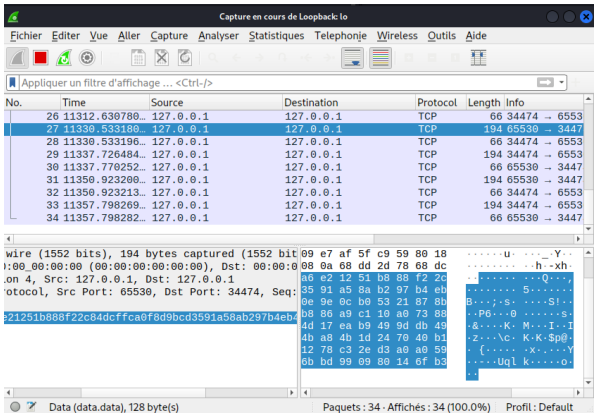


**Figure 4.26 :** premier message crypté de client 1 « hi ».

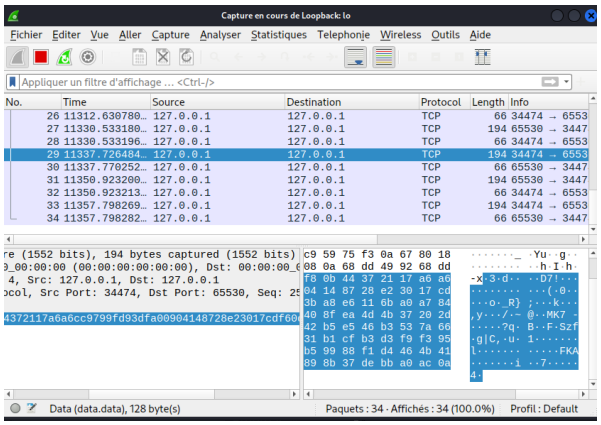


**Figure 4.27 :** premier message crypté de client 2 « hello ».

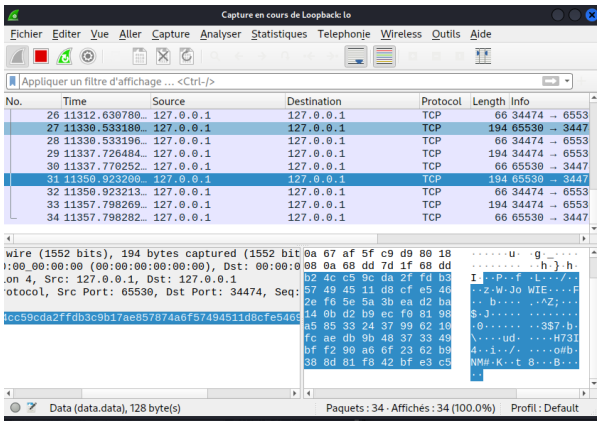


**Figure 4.28 :** deuxième message crypté de client 1 « how are you? ».

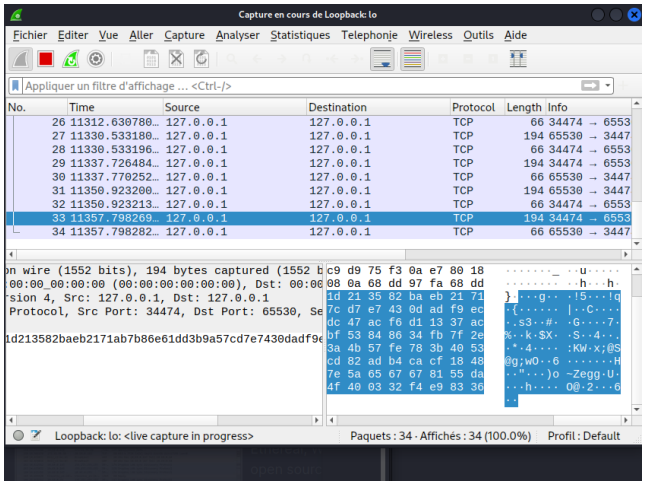


**Figure 4.29 :** deuxième message crypté de client 2 « fine ».

## 4.5 Conclusion

Dans ce chapitre, nous avons testé les codes Python, et les résultats ont confirmé que la cryptographie est une méthode efficace pour garantir la sécurité des informations lors des communications. Nous avons également constaté qu'il existe plusieurs procédures pour appliquer cette technique, qui dépendent des fonctions de sécurité telles que la confidentialité et l'intégrité.

En utilisant des algorithmes tels que RSA, Diffie-Hellman, le chiffrement simple et le hachage SHA1, nous avons pu mettre en place des systèmes de cryptographie robustes. Ces systèmes permettent de protéger les données sensibles et de garantir la confidentialité et l'intégrité des informations échangées.

La cryptographie joue un rôle crucial dans la sécurité des communications et continue d'évoluer pour faire face aux nouveaux défis de notre ère numérique. Il est essentiel de comprendre et de mettre en pratique ces concepts pour assurer la protection des données dans nos échanges quotidiens.

# CONCLUSION GENERALE

Dans le cadre de notre mémoire, nous avons abordé les problématiques liées à la sécurité de l'information sensible. À travers notre étude, nous avons implémenté et testé divers algorithmes de chiffrement et d'intégrité, ce qui a permis d'améliorer le niveau de sécurité des données.

Nous pouvons conclure qu'il existe plusieurs fonctions de cryptographie qui dépendent des fonctions de sécurité. La cryptographie se révèle être une méthode efficace, mais elle présente également certaines limites. Par exemple, la cryptographie à clé secrète est limitée en cas de clé distribuée ou partagée entre plusieurs personnes. De plus, la cryptographie à clé publique nécessite un nombre important de calculs et l'utilisation de différentes clés pour le chiffrement et le déchiffrement, ce qui peut entraîner des temps de calcul très longs. Un autre défi majeur réside dans l'échange sécurisé des clés lors de la transmission de données chiffrées. Il est primordial de s'assurer que cet échange se fait de manière confidentielle.

Malgré ces défis, la cryptographie reste un outil essentiel pour assurer la sécurité des données sensibles. Elle joue un rôle crucial dans la protection des informations confidentielles lors des communications et des transactions en ligne. Il est important de rester à jour avec les avancées de la cryptographie et de choisir les méthodes les plus adaptées en fonction des besoins spécifiques de sécurité.

En conclusion, notre travail de recherche nous a permis de mieux comprendre les enjeux de la sécurité de l'information sensible et de mettre en pratique des techniques de cryptographie pour renforcer la protection des données. La sécurité de l'information demeure un domaine en constante évolution, et il est essentiel de rester vigilant et de continuer à explorer de nouvelles solutions pour faire face aux défis de sécurité actuels et futurs.

# Bibliographie

[1] A. Makhloufi « Sécurité de l'information ». Support de cours destiné aux étudiants de Licence L3 SIE. Université de Batna 2, Algérie. (2019/2020).

[2] F. Bergeron et A. Goupil « La cryptographie de l'Antiquité à l'internet ». Université du Québec à Montréal, Canada. (28 avril 2014).

[3] B. Collard «La cryptographie dans l'Antiquité gréco-romaine». Université de Louvain à Bruxelles, Belgique. (janvier-juillet 2004).

[4] K. Stiti et S. Bellouni « Sécurité des données par la cryptographie quantique ». Mémoire de master. Université de Mouloud Mammeri à Tizi-Ouzou, Algérie. (2010/2011).

[5] D. Lamas « Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES ». Haute Ecole de Gestion de Genève (HEG-GE). 5 juin 2015.

[6] Ziani Rezki « Cryptographie et sécurité des Réseaux Implémentation de l'AES sous MATLAB ». Mémoire de fin d'études, Université MOULOUD MAMMERI, Tizi-Ouzou, Algérie. (2008).

[7] Dr N. Bounour « La cryptographie : principe et méthodes ». Cour de master 1 ILC, 2020/2021.

[8] J. Blanc « Techniques de cryptographie ». Licence Informatique. (2003/2004).

[9] J-P. Gaulier « Advanced Encryption Standard : Présentation de l'AES ». Mémoire de synthèse, Conservatoire national des arts et métiers, Paris, France. (2004).

[10] S. Jacob « Protection cryptographique des bases de données : Conception et Cryptanalyse ». Cryptographie et Sécurité [cs.CR]. Université Pierre et Marie Curie, Paris, France. (2012).

[11] D. Pointcheval « La cryptographie Asymétrique et les Preuves de Sécurité ». Chargé de recherche CNRS, Département d'informatique, Ecole normale supérieure. (2002).

[12] Y. Legrandgérard « Cours de cryptographie». Université Paris Diderot, Laboratoire IRIF. (Septembre 2019).

[13] D. Barsky et G. Dartois « Cryptographie ». Cours, Paris, France. (2010).

[14] Z. Dahmane et L. Abdelli « Implémentation d'un algorithme de cryptage sur un circuit FPGA ». Mémoire de master, Université Mohamed Boudiaf, M'sila, Algérie. (mai 2017).

[15] « Les Fonctions de Hachage ». ÉPREUVE COMMUNE DE TIPE 2008 - Partie D.

[16] Mahdi Hamza « étude et comparaison des principaux systèmes de cryptage et les techniques y afférentes ». Mémoire de master, Université de M'sila, Algérie. (2015/20 16).

[17] Z-I. Lotmani et Y. Elhomr « Simulation d'une attaque sur le crypto système RSA ». Mémoire de master, Université Abdelhamid Ibn Badis, Mostaganem, Algérie. (2013/2014).

[18] A. Miroud et M. Nouadri « Cryptanalyse de RSA : Etude comparative de deux approches». Mémoire de master, Université Larbi Ben M'Hidi, Oum El Bouaghi, Algérie. (juin 2016).

[19] N. Lebsir « CHARM : Crypto système Hybride Avancé pour les Réseaux Mobiles ». Mémoire de master, Université Mohammed Seddik Ben Yahia, Jijel, Algérie. (2019/2020).

[20] K. Djebaili « Méthodes de chiffrement basés sur la factorisation en entiers et logarithme discret ». Thèse : Doctorant en Informatique, Université de Batna 2, Algérie. (16/02/2017).

[21] S. Hamzaoui « Techniques de Cryptographie ». Mémoire de magister en Mathématique, Université des Sciences et de technologie Houari Boumediene, Bab Ezzouar, Algérie. (05/01/2004).

[22] B. MEZIANI et M. OUAHIANI « Sécuriser la communication de groupe dans les réseaux VANETs « protocole Diffie-Hellman » ». Mémoire de master, Université de Tlemcen, Algérie. (2016/2017).

[23] A. MAMERI et B. OUBELLIL « Conception et réalisation d'un système de hachage basé sur la fonction SHA-1 ». Mémoire de master, Université Mouloud Mammeri, Tizi-Ouzou, Algérie.

[24] Madalina Fantana « Algorithmes de hachage : MD5 et SHA 1 ». Cours, Technical University of Cluj-Napoca, Romania. (Janvier 2016).

[25] P. Fuchs et P. Poulain « Cours de Python ». Cours. Université Paris Cité, France. Version du 29 août 2022.

[26] MOUMOUNI Boubacar Moussa et DAN BAKI ISSAKA Mahamadou Sanoussi « Développement d'une solution complète pour le service pédiatrie B - CHU Tlemcen ». Mémoire de master, Abou Bakr Belkaid. Tlemcen, Algérie. (27 juin 2019).