UNIVERSITE KASDI MERBAH OUARGLA

Faculté des Nouvelles Technologie d'Information et de la communication

Département DE L'INFORMATIQUE ET TECHNOLOGIES DE L'INFORMATION

Mémoire

MASTER ACADEMIQUE

Domaine : Mathématiques et Informatiques

Filière : Informatique

Spécialité : **Informatique Fondamentale**

Présenté par : DEBBAGH BESMA et BOUNEGEB NOURA

Thème

Eude et comparaison de principaux systèmes crypto Fournis par le package de Bouncy Castle Plate forme Java SDK

Soutenu publiquement

le : 05/06 /2016

Devant le jury :

Mr BEKKARI Fouad	MA (A)	Président	UKM Ouargla
Mr. EUSCHI Salah	MA (A)	Encadreur/rapporteur	UKM Ouargla
Mr. MAHDJOUBE Med Bachir	MA (A)	Examinateur	UKM Ouargla

Année Universitaire : 2015 /2016

Remerciement

Pour louer et remercier Dieu d'abord , avancer sincères remerciements et sa gratitude au superviseur en faveur du Encadreur : Salah Euschi de nous accorder le temps et l'effort et de l'orientation et des conseils, et d'offrir grâce du professeur Sami Lemtannech pur et profond au bout et les efforts qui nous ont donné ,et le messiéent président de juré Mr BEKKARI Fouad le messiéent membre de juré Mr. MAHDJOUBE Med Bachir et nous remercions tous valorisés mes professeurs et à la fois contribué au tutoriel.et tout la famille debbagh et bounegab et tout les amis.

Dédicace

Je dédie ce mémoire à :

Mes parents :

Ma mère, qui a œuvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie, reçois à travers ce travail aussi modeste soit-il, l'expression de mes sentiments et de mon éternelle gratitude.

Mon père, qui peut être fier et trouver ici le résultat de longues années de sacrifices et de privations pour m'aider à avancer dans la vie. Puisse Dieu faire en sorte que ce travail porte son fruit ; Merci pour les valeurs nobles, l'éducation et le soutient permanent venu de toi .

Mes frères et sueurs et ma famille qui n'ont cessé d'être pour moi des exemples de persévérance, de courage et de générosité.

Pour l'esprit de Grande mère

Le professeur ***SALAH EUSCHI***

Le Frère et professeur SAMI LEMTANNECH

Ma sœur et mon collègue ***BESSMA***

Mes amies et tout les étudiants de masters informatiques surtout les étudiant de informatique fondamental

Mes professeurs qui doivent voir dans ce travail la fierté d'un savoir bien acquis.

NOURA

Dédicace

Je dédie ce mémoire à :

Mes parents :

Ma mère, qui a ouvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie, reçois à travers ce travail aussi modeste soit-il, l'expression de mes sentiments et de mon éternelle gratitude.

Mon père, qui peut être fier et trouver ici le résultat de longues années de sacrifices et de privations pour m'aider à avancer dans la vie. Puisse Dieu faire en sorte que ce travail porte son fruit ; Merci pour les valeurs nobles, l'éducation et le soutient permanent venu de toi.

Mes frères et sueurs et ma famille qui n'ont cessé d'être pour moi des exemples de persévérance, de courage et de générosité.

Pour l'esprit de ma tante précieuse ***Samira***

Le professeur ***Salah Euschi***

Ma sœur et mon collègue ***Noura***

Mes amies et tout les étudiants de masters informatiques surtout les étudiant de informatique fondamental

Mes professeurs qui doivent voir dans ce travail la fierté d'un savoir bien acquis.

BESSMA

Table des matières

Liste des figures …… 4

Liste des tables ……..…… 4

Abbréviation ……...…… 5

Introduction Générale …… 6-7

Problématique ……...……...…….…...8

Chapitre I : Concepts et Principes De La Cryptographie

I.1 Introduction ……...…….…… 10

I.2 Histoire de la cryptographie ……...……..……10- 11

I.3 Définition et terminologie ……..…….....……..……..12-13

I.4 Grands Principes de la cryptographie ……...……...……..……13

I.5 Objectif de la cryptographie …….....……..……..…13-14

I -6 La cryptographie classique ……...……..…….……14

I.6.1 le chiffrement par décalage ……..…...…….…….....…14-15

I.6.2 Le chiffrement par substitution ……...……...…….…….……15

I.6.2.1 La substitution mono alphabétique ……...……..…….……..…15

I.6.2.2 La substitution poly alphabétique …….…….…..…15

I.6.2.3 La substitution homophonique …….....…….…...……15

I.6.2.4 La substitution de poly grammes ……..……..…….…..…15

I.6.3 Chiffrement par transposition ou chiffrement par permutation ...…….……. 16

Chapitre II: Les Concepts Fondamentaux De La Cryptographie modern

II .1 Mécanismes de la cryptographie…….……....……...……..18

II -2 La cryptographie moderne …….……18

II.3 Principes de la cryptographie moderne ……...……...…..…18-19

II.2.1 La cryptographie symétrique ……...……...……..... 19

II.2.1.1 Principe général …….....……19

II.2.1.2 Chiffrement par bloc (Block Cipher) ……...……..……20

II.2.1.3 Chiffrements de flux (Stream Cipher) ……...…….…...…22

II.2.1.4 Considérations de sécurité pour la cryptographie symétrique……..…22

II.2.1.5 Gestion de clé ……..……...…….…22

II.2.1.6 Authentifier le chiffrement symétrique ……...…..……23
II.2.2 Cryptographie asymétrique ……...…….…….……23
II.2.2.1 Problème de la factorisation ……..……..….……24
II.2.2.2 Cryptage RSA ……...……..…….……..…25
II.2.2.3 Considérations de sécurité pour l'utilisation de RSA …...…….…… 27
II.2.2.4 Problème Logarithme discret ……...……...……28
II.2.2.5 Autres algorithmes Asymétrique ……...……..……28
II.2.2.6 Problème du Logarithme Discret pour les courbes elliptiques …..…. 30
II.2.3 les avantages et les Inconvénients de la cryptographie symétrique et asymétrique…..31
II .2.4 Les systèmes cryptographiques hybrides ou mixtes …….…...……. 31
II.2.5 Fonctions de hachage ……...……...……...…… 33
II.2.5.1 Algorithme de hachage sécurisé SHA ……...……..……33
II.2.5.2 Autres algorithmes de hachage ……...……...……..…34
I I.2.6 Code d'authentification de message MAC ……...……...…34
II.2.6.1 HMAC ……...……..……34
II.2.6.2 Autres fonctions MAC ……..…….….…35
II.2.7 Certificats à clé publique et les signatures numériques ……..……..…… 35
II.2.7.1 Certificats à clé publique et le format X.509 …...……..… 36
II.2.7.2 Signatures numériques……...…… 37
Chapitre III : La cryptographie dans java et la package cryptographique de Bouncy Castle
III.1 Introduction ……...…….……39
III.2 Sécurité du langage Java ……...……..…….……39
III.3 Cryptographie dans Java …….....……40
III.3.1 Concepts cryptographiques implémentés par l'API de sécurité Java .....….. 40
III.3.2 Services cryptographique fournis …….….41
III.4 Architecture JCA/JCE ……...……..……41
III.4.1 Présentation des architectures JCA et JCE ……...…….. 41
III.4.1.1 JCA (Java Cryptographie Architecture) …….… 41
III.4.1.2 JCE (Java Cryptographie Extension) ...…….…...… 42
III.4.2 Principes de conception des providers dans JCA/JCE……...43
III.5 Les concepts principaux introduits dans l'API de sécurité ..... …….…...… 44
III.5.1 Classes de moteur et les algorithmes ……...……..…….……44
III.5.2 Implémentations et Providers ……...……..…….……45
III.6 La classe Provider …….....……45

III.6.1 Comment utiliser les implémentations d'un provider ? …….…….. 45
III.6.2 Installation et configuration de providers ……..……. 46
III.7 Le Provider De Bouncy Castle ……..…..……...…47
III.7.1 Présentation du package crypto de Bouncy Castle (BC) ……..…. 47
III.8 L'environnement de développement intégré NetBeans ...…….....…….……..…..…48
Chapitre IV : Etude et comparaissent Des algorithmes crypto fournis par le package de Bouncy Castle
IV .2 Le Test de Bouncy Castle …….…….……..…..… 51
IV .2 Algorithmes de chiffrement symétriques …….……..……..…… 51
IV .2 .1 Comparé le tempe d'exécution de Chiffré et Déchiffré les Fichiers différent tailles en utilisant les algorithmes de Symétrique dans la Cryptage moderne …..……. 52
IV .2 .2 Comparé le tempe d'exécution de Chiffré et Déchiffré les Textes en utilisant les algorithmes de Symétrique dans la Cryptage moderne ……...……..52
IV .3 Algorithmes de chiffrement Asymétriques ……...…….……53
IV.3 Fonctions de hachage et Macs ……..…… 54
Conclusion générale : ……...…….…… 55
Référence:……...……..……56
Résumé

Liste des figures :

Figure 1. Schéma général de la cryptologie ……8
Figure 2 . Chiffrement et Déchiffrement ……9
Figure I.1 . Illustration- Nomenclature ……11
Figure I.2 : Chiffrement / déchiffrement ……12
Figure I.2 . les chiffrement par décalage ……15
Figure I.3 . les chiffrement par substitution ……16
Figure II.1 . Les systèmes de chiffrement ……18
Figure II.2 : Chiffrement symétrique ……19
Figure II.3 : Algorithme TDES ……21
Figure II.4 : Chiffrement Asymétrique ……24
Figure II.5 : Chiffrement hybrides ……32
Figure II.6 : Certificat X.509 ……37
Figure III.1 : L'API JCA/JCE et les providers de service ……42
Figure IV.1: Interface principale ……50
Figure IV.2: Interface secondaire ……50
Figure IV.3 Test de Bouncy Castle ……51
Figure IV.4: Algorithmes symétriques ……51
Figure IV.5: Algorithmes Asymétrique ……53
Figure IV.6: Signature numérique ……53
Figure IV.7: Génération de Pair de clé : algorithmes RSA ……53
Figure IV.8: Fonction de Hachage ……54

Liste des tables :

Table II.1 : Les applications des crypto systèmes asymétriques …… 24
Table II.2 : les avantages et les Inconvénients symétrique / asymétriques ……31
Table IV .1 Comparaissent les fichiers ……52
Table IV .2 Comparaissent les textes ……52

***Abbreviation* :**

A

AES Advanced Encryption Standard
ANSI American National Standards Institute
API Application Programmation Interface

B

BC Bouncy Castle

C

CA Certification Authority
CBC Cipher-Block Chaining mode of operation
CFB Cipher Feedback mode of operation
CPU Computer Processing Unite
CRL liste de révocation de certificat
CRT Chinese Remainder Theorem
CSP Cryptographic Service Provider

D

DES Data Encryption Standard
DH Diffie-Hellman
DL Discrete Logarithm
DLP Discrete Logarithm Problem
DSA Digital Signature Algorithm
DSS Digital Signature Standard

E

ECB Electronic Codebook mode of operation
ECC Elliptic Curve Cryptography
ECDH Elliptic Curve Diffie-Hellman
ECDSA Elliptic Curve Digital Signature Algorithm
ECIES Elliptic Curve Integrated Encryption Scheme
EDI Environnement de Développement Intégré

F

FIPS Federal Information Processing Standards

G

GF(p) Corps fini (Galois Field) premier à p éléments
GF(2n) Corps fini (Galois Field) binaire à 2n éléments

S

SEC Standards for Efficient Cryptography
SECG Standards for Efficient Cryptography Group
SHA-1 Secure Hash Algorithm 1
SSL Secure Sockets Layer
SPI Service Provider Interface

H

HMAC Hash-based Message Authentication Code

I

IBM International Business Machines
IDEA Norme internationale de cryptage de données
IEEE Institute of Electrical and Electronics Engineers
IPSec Internet Protocol Security
ISO Organisation internationale de normalisation
IV Initialization Vector

J

JCA Java Cryptography Architecture
JCE Java Cryptography Extension
JRE Java Runtime Environnement
JVM Machine Virtuelle Java

K

KDF Key Derivation Function

M

MAC Message Authentication Code
MD4 Message Digest 4
MD5 Message Digest 6

N

NIST National Institute of Standards and Technology
NSA National Security Agency

P

PBE Password Based Encryption
PKCS Public Key Cryptography Standards
PKI Public-Key Infrastructure
PRNG Random Number Generator

R

RC2 Rivest Cipher 2
RC4 Rivest Cipher 4
RSA Rivest-Shamir-Adleman

T

TLS Transport Layer Security
TDES Triple Data Encryption Standard

W

WEP Wired Equivalent Privacy

X

XOR eXclusive OR

Introduction Général :

Suite au développement rapide ces dernières années des technologies de l'information et la communication et les infrastructures réseaux filaires et sans fil, la cryptographie est devenue, depuis quelques décennies, un véritable enjeu de société. Le développement rapide des réseaux de communication numériques, ainsi que l'augmentation du nombre de ses utilisateurs, ont posé de façon de plus en plus critique le problème de l'échange des clés de chiffrement. Très schématiquement, le principe de la cryptographie classique consiste à échanger des données chiffrées à partir d'un secret (appelé clé) connu uniquement par les parties concernées, le chiffrement comme le déchiffrement nécessitant la clé. Le paradoxe étant que pour pouvoir échanger des informations secrètement, les deux parties doivent déjà partager un secret, le problème d'échange de clés prend une tout autre dimension à l'heure d'internet et de son milliard d'usagers. C'est pour répondre à ce problème qu'a été inventée la cryptographie à clé publique, le principe de la cryptographie à clé publique est d'utiliser non plus une clé pour chiffrer et déchiffrer mais un couple de clés : une clé publique, donc connue de tous, servant au chiffrement, et une clé privée, connue uniquement du destinataire, permettant le déchiffrement. Ainsi si une personne a besoin de se faire envoyer des informations de façon confidentielle, il lui suffit de générer elle même un couple clé publique clé privée et plus aucun échange de clés n'est alors nécessaire (mais se pose alors le problème de la diffusion de cette clé publique).Si le principe de la cryptographie à clé publique fut proposé en 1975, ce n'est qu'en 1977 que fut présenté le premier protocole effectif : RSA. Principalement basé sur le problème de la factorisation des grands entiers, RSA est encore.

Aujourd'hui le crypto système le plus utilisé en cryptographie. Cependant les nombreux progrès effectués dans le domaine de la factorisation font que la taille des clés RSA augmente plus vite que ne le requiert l'augmentation de la puissance des ordinateurs. C'est l'une des raisons pour lesquelles la cryptographie basée sur les courbes elliptiques (ECC) connaît un tel intérêt depuis son introduction par Miller et Ko blitz en 1985.

Les systèmes cryptographiques basés sur les courbes elliptiques permettent d'obtenir un gain en efficacité dans la gestion de clés. En effet, de tels crypto systèmes nécessitent des clés de taille beaucoup plus modeste, par exemple, une clé ECC de 160 bits réalise le

même niveau de sécurité qu'une clé RSA 1024 bits, ce qui représente un avantage pour les systèmes utilisant les cartes à puces dont l'espace mémoire est très limité.

De plus, les algorithmes de calculs liés aux courbes elliptiques sont plus rapides, et ont donc un débit de générations et d'échanges de clé beaucoup plus important.

L'objectif de notre travail consiste à réaliser des benchmarks pour mesurer les opérations crypto, Ces benchmarks de comparaison nous aident dans le choix des algorithmes exacts et les plus appropriées à notre environnement matériel et logiciel utilisé. Nous avons choisi d'utiliser le package crypto de Bouncy Castle sous la plateforme Java J2SE, parce qu'il est open source, riche en algorithmes crypto et implémente des classes implémentant Notre mémoire est organisé en quatre chapitres, dans le premier chapitre nous présenterons les définitions de la cryptographie moderne comme ils sont définis par les standards internationaux de la sécurité, nous étudierons, dans le deuxième chapitre nous montrons des concepts fondamentaux et certains concepts mathématiques relatives ensuite nous présenterons dans le troisième chapitre l'environnement pratique de notre travail qui consiste à la plateforme Java avec son architecture de cryptographie JCA/JCE et le provider JCE de Bouncy Castle, et enfin nous étudierons l'implémentation de benchmarks d'opérations crypto et la comparaison de ces algorithmes .

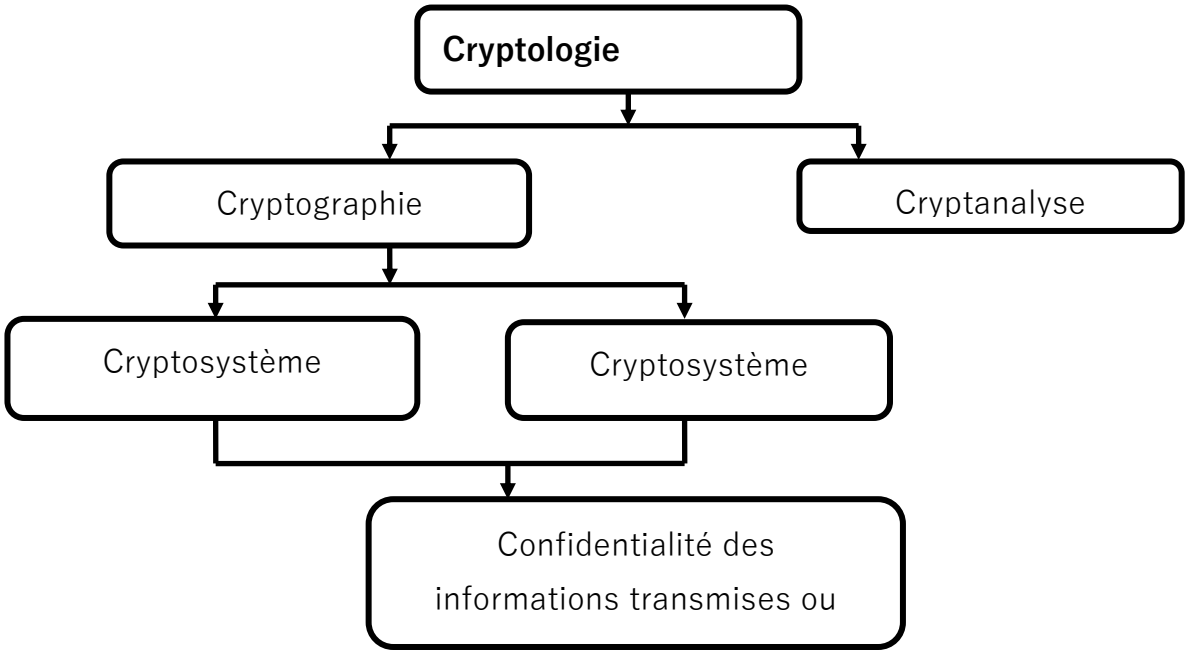

Figure 1. Schéma général de la cryptologie

Problématique :

La cryptographie permet de sécuriser les communications et fournir l'authentification, mais l'application de la cryptographie au niveau logiciel dans les applications web peut ralentir le temps d'exécution. Si la cryptographie n'est pas bien implémentée, on pourra avoir un problème de performance. Un mauvais choix des algorithmes crypto et des tailles de clés peuvent affaiblir la sécurité du système. Pour assurer la non-répudiation, une infrastructure de certificats à clé publique tierce partie doit être utilisée. La PKI (Public Key Infrastructure) est le système leader de la sécurité des réseaux et des transactions e-commerce. Elle permet à des utilisateurs d'échanger des données et de l'argent d'une manière sécurisée et confidentielle.

Les développeurs d'applications web ont besoins d'algorithmes de cryptographie. On trouve ces algorithmes enveloppés dans un ensemble de packages qui offrent des services cryptographiques aux développeurs. Le package est une API qui permet au développeur de sécuriser le contenu de données au lieu de sécuriser uniquement la connexion avec le serveur. Pour sécuriser le contenu des données, les mécanismes de sécurité devront être appliqués au niveau de la couche application .

Figure 2 . Chiffrement et Déchiffrement

Chapitre I :

Concepts et principes De La Cryptographie

I.1 Introduction :

Notre projet rentre dans le cadre de la sécurité informatique. La cryptographie n'est pas toute la sécurité mais nous ne pouvons pas réaliser la sécurité informatique sans cryptographie.

La cryptographie permet de réaliser les objectifs de sécurité de confidentialité intégrité authentification, et de disponibilité via ses outils de chiffrement, signature fonctions de hachage et Mac (Code d'authentification de message).

La cryptographie permet de sécuriser les communications et fournir l'authentification, mais l'application de la cryptographie au niveau logiciel dans les applications web peut ralentir le temps d'exécution. Si la cryptographie n'est pas bien implémentée, on pourra avoir un problème de performance. Un mauvais choix des algorithmes crypto et des tailles de clés peuvent affaiblir la sécurité du système. Pour assurer la non-répudiation, une infrastructure de certificats à clé publique tierce partie doit être utilisée. La PKI(Public Key Infrastructure) est le système leader de la sécurité des réseaux et des transactions e-commerce. Elle permet à des utilisateurs d'échanger des données et de l'argent d'une manière sécurisée et confidentielle .

Les développeurs d'applications web ont besoins d'algorithmes de cryptographie. On trouve ces algorithmes enveloppés dans un ensemble de packages qui offrent des services cryptographiques aux développeurs. Le package est une API qui permet au développeur de sécuriser le contenu de données au lieu de sécuriser uniquement la connexion avec le serveur. Pour sécuriser le contenu des données, les mécanismes de sécurité devront être appliqués au niveau de la couche application.

I .2 Histoire de la cryptographie

Dès l'Antiquité, les peuples employèrent des codes secrets dans certains de leurs textes les archéologues en ont découvert dans des hiéroglyphes égyptiens et dans des écrits cunéiformes (écriture mésopotamienne). De même, les Hébreux dissimulaient parfois leurs écrits en inversant l'alphabet, c'est-à-dire en employant la dernière lettre de l'alphabet à la place de la première, l'avant-dernière lettre à la place de la deuxième, et ainsi de suite.

Les applications militaires de la cryptographie furent nombreuses au cours des siècles. Durant la Seconde Guerre mondiale, les Allemands communiquèrent ainsi par radio grâce au code Enigme, que les Britanniques réussirent à percer grâce à un ordinateur numérique baptisé Colosses.

De la même manière, les Américains parvinrent à décrypter certains chiffres employés par les Japonais.

Ces divers succès, qui contribuèrent pour beaucoup à la victoire des Alliés, ont prouvé qu'aucun cryptage, aussi sophistiqué soit-il, ne doit être considéré comme inviolable, et qu'il est par conséquent dangereux de lui faire aveuglement confiance. Les ordinateurs et le réseau Internet font entrer la cryptologie dans son ère moderne. La grande invention de ces dernières décennies fut la cryptographie à clés publiques. Au début des années 1970, Horst Feistel a mené un projet de recherche à IBM Watson Research Lab. qui a développé le chiffre Lucifer, qui inspira plus tard le chiffre DES et d'autres chiffres.

En 1976, Winfield Diffie et Martin Hellman publient « New Directions in cryptography » introduisant l'idée de cryptographie à clef publique. Ils donnent une solution entièrement nouvelle au problème de l'échange de clés.

En novembre 1976, DES est un algorithme très répandu à clé privée dérivé du chiffre Lucifer de Feistel (de chez IBM) dans sa version à 64 bits.

Avril 1977, l'apparition du fameux crypto système RSA, le brevet de cet algorithme appartient à la société américaine RSA Data Security. RSA est un algorithme à clé publique qui sert aussi bien à la cryptographie de documents, qu'à l'authentification.

En 1978, l'algorithme RSA est publié dans les Communications de l'ACM.

En 1985, Victor Miller et Neal Koblitz utilisent les courbes elliptiques pour la cryptographie.

[11]

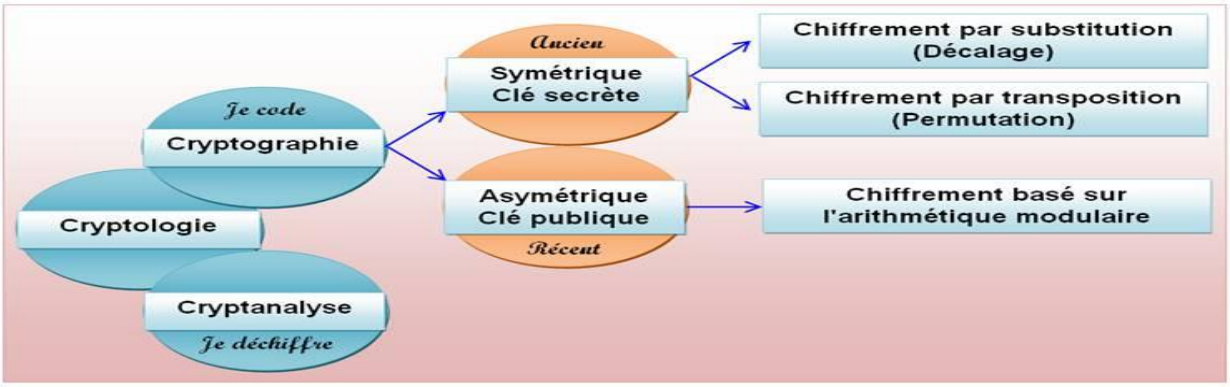

Figure I.1 . Illustration- Nomenclature

I .3 Définition et terminologie

Définition de la cryptographie :

La cryptographie est l'art de rendre inintelligible, de crypter, de coder un message à ceux qui ne sont pas habilités à en prendre connaissance, c'est est un ensemble des principes, méthodes et techniques dont l'application assure le « crypter » et le « décrypter » des données , cette branche regroupe l'ensemble des méthodes qui permettent de chiffrer et de déchiffrer un texte en clair afin de le rendre incompréhensible pour quiconque n'est pas en possession de la clé à utiliser pour le déchiffrer.

La signification de « crypter » et « décrypter » est donnée successivement comme suit:

Crypter: brouiller l'information, la rendre "incompréhensible"

Décrypter: rendre le message compréhensible

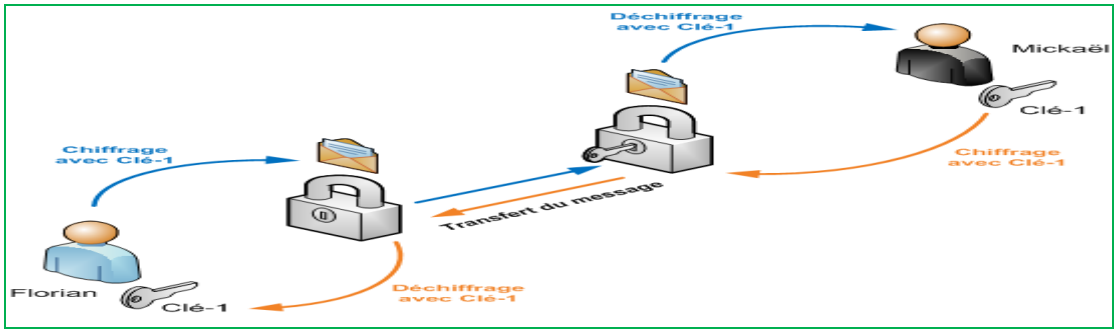

Figure I.2 : Chiffrement / déchiffrement

Texte en clair : c'est le message à protéger.

Texte chiffré : c'est le résultat du chiffrement du texte en clair.

Déchiffrement c'est la méthode ou l'algorithme utilisé pour transformer un texte chiffré en texte en clair

Clé : c'est le secret partagé utilisé pour chiffrer le texte en clair en texte chiffré et pour déchiffrer le texte chiffré en texte en clair. On peut parfaitement concevoir un algorithme qui n'utilise pas de clé, dans ce cas c'est l'algorithme lui-même qui constitue la clé, et son principe ne doit donc en aucun cas être dévoilé.

Cryptanalyse : c'est l'art de révéler les textes en clair qui ont fait l'objet d'un chiffrement sans connaître la clé utilisée pour chiffrer le texte en clair.

Cryptologie : il s'agit de la science qui étudie les communications secrètes. Elle est composée de deux domaines d'étude complémentaires : la cryptographie et la cryptanalyse.

Cryptosystème : est constitué d'un algorithme cryptographique ainsi que toutes les clés possibles et tous les protocoles qui le font fonctionner**.**

Crypter : en relisant la définition du mot décrypter, on peut se rendre compte que le mot crypter n'a pas de sens et que son usage devrait être oublié. Le mot cryptage n'a pas plus de sens non plus.

Décrypter : c'est l'action de retrouver le texte en clair correspondant à un texte chiffré sans posséder la clé qui a servit au chiffrement. Ce mot ne devrait donc être employé que dans le contexte de la cryptanalyse.

Coder/ Décoder : c'est une méthode ou un algorithme permettant de modifier la mise en forme d'un message sans introduire d'élément secret .

I .4 Les grands principes de la cryptographie :

Le cryptage est une opération qui consiste à transformer un message clair en un message indéchiffrable pour tous, sauf pour les destinataires du message original (qui peut être l'émetteur lui-même). Il sert donc le plus souvent à prévenir une éventuelle interception par un tiers. On utilise pour cela un algorithme et une ou plusieurs clés, secrètes ou publiques L'émetteur crypte son message à l'aide d'un algorithme, le transmet crypté, et l'émetteur peut alors le décrypter à l'aide d'une même clé ou d'une différente . Pourquoi utiliser le cryptage des informations **?[12]**

I .5 Les objectifs de la cryptographie

Les principaux objectifs à garantir par l'application de la cryptographie sont:

- **Confidentialité:** un mécanisme pour transmettre des données de telle sorte que seul le destinataire autorisé puisse les lire.

- **Intégrité des données:** un mécanisme pour s'assurer que les données reçues n'ont pas été modifiées durant la transmission, frauduleusement ou accidentellement.
- **Authentification:** un mécanisme pour permettre d'authentifier les utilisateurs de façon à limiter l'accès aux données, serveurs et ressources aux seules personnes autorisées (un mot de passe par un nom de login ou un certificat numérique).
- **Non-répudiation:** un mécanisme pour enregistrer un acte ou un engagement d'une personne ou d'une entité de telle sorte que celle-ci ne puisse pas nier avoir accompli cet acte ou pris cet engagement. Ce mécanisme se décompose:
  - non-répudiation d'origine l'émetteur ne peut nier avoir écrit le message.
  - non-répudiation de réception, le receveur ne peut nier avoir reçu le message.
  - non-répudiation de transmission, l'émetteur du message ne peut nier avoir envoyé le message. **[13]**

I .6 La cryptographie classique

Cette partie introductive a pour but de décrire dans ses grandes lignes le paysage cryptologique à travers l'histoire de l'humanité, le paysage cryptologique actuel est le fruit d'une longue évolution, qui a vu la cryptologie passer progressivement de l'état de technique artisanale à celui de science.

Au cours de cette évolution, plusieurs objectifs se sont dessinés pour cette discipline : l'intégrité, l'authenticité, la confidentialité par ailleurs le développement massif des technologies de l'information a mis en évidence de nouvelles exigences de sécurité et permis l'émergence de nouveaux scénarios cryptologiques.

Plusieurs types de chiffrement:

- chiffrement par décalage
- chiffrement par substitution
- chiffrement par transposition ou chiffrement par permutation .

I .6.1 Le chiffrement par décalage :

Le chiffrement par décalage, aussi connu comme le chiffre de César ou le code de César (voir les différents noms), est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes (ce qui explique le nom « chiffre de César »).

Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet. Pour les dernières

lettres (dans le cas d'un décalage à droite), on reprend au début. Par exemple avec un décalage de 3 vers la droite, A est remplacé par D, B devient E, et ainsi jusqu'à W qui devient Z, puis X devient A etc. Il s'agit d'une permutation circulaire de l'alphabet. La longueur du décalage, 3 dans l'exemple évoqué, constitue la clé du chiffrement qu'il suffit de transmettre au destinataire — s'il sait déjà qu'il s'agit d'un chiffrement de César — pour que celui-ci puisse déchiffrer le message. Dans le cas de l'alphabet latin, le chiffre de César n'a que 26 clés possibles (y compris la clé nulle, qui ne modifie pas le texte).

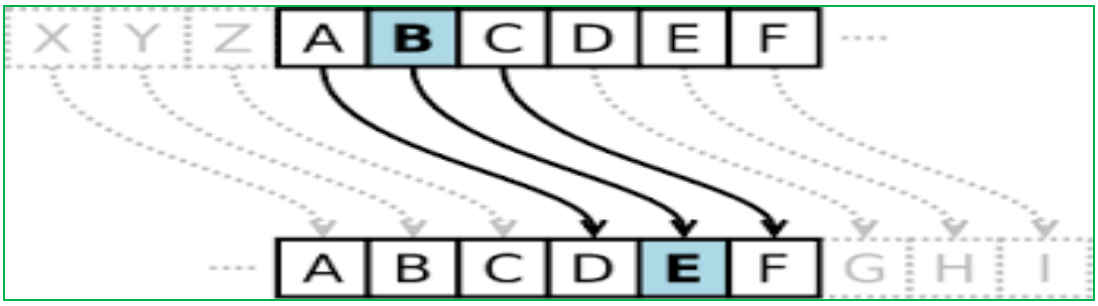

Figure I.3 . les chiffrement par décalage

I .6.2 Le chiffrement par substitution

Le chiffrement par substitution consiste à remplacer dans un message une ou plusieurs entités (généralement des lettres) par une ou plusieurs autres entités.

On distingue généralement plusieurs types de crypto systèmes par substitution:

I .6.2.1 La substitution mono alphabétique :

Consiste à remplacer chaque lettre du message par une autre lettre de l'alphabet.

I .6.2.2 La substitution poly alphabétique :

Consiste à utiliser une suite de chiffres mono alphabétique réutilisée périodiquement

I .6.2.3 La substitution homophonique:

Permet de faire correspondre à chaque lettre du message en clair un ensemble possible d'autres caractères.

I .6.2.4 La substitution de poly grammes :

Consiste à substituer un groupe de caractères (poly gramme) dans le message par un autre groupe de caractères.

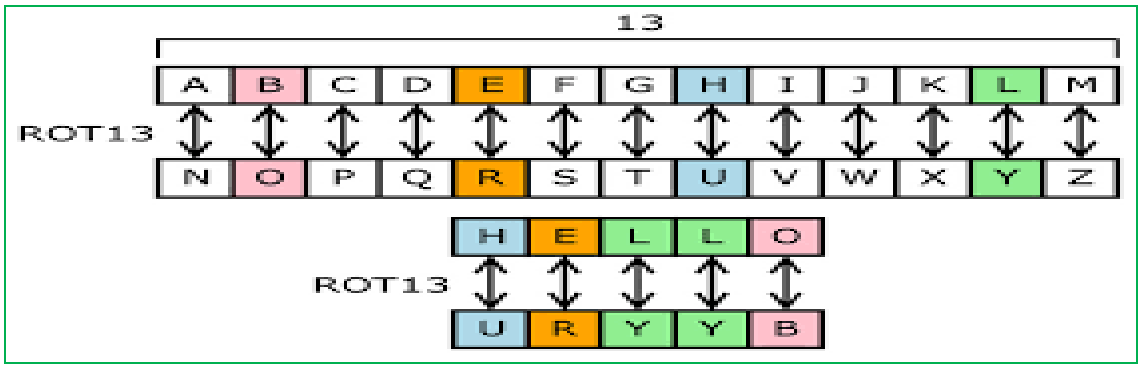

Figure I.4 . Les chiffrements par substitution

I .6.3 Chiffrement par transposition ou chiffrement par permutation

Les méthodes de chiffrement par transposition consistent à réarranger les données à crypter de telle façon à les rendre incompréhensibles. Il s'agit généralement de réarranger géométriquement les données pour les rendre visuellement inexploitables.**[14]**

Chapitre III :

Les Concepts Fondamentaux De La Cryptographie moderne

II .1 Mécanismes de la cryptographie :

Un algorithme de cryptographie ou un chiffrement est une fonction mathématique utilisée lors du processus de cryptage et de décryptage. Cet algorithme est associé à une clef (un mot, un nombre, ou une phrase). Afin de crypter une donnée avec des clés différentes le résultat du cryptage variera également. La sécurité des données cryptées repose entièrement sur deux éléments : l'invulnérabilité de l'algorithme de cryptographie et la confidentialité de la clef .

Un système de cryptographie est constitué d'un algorithme de cryptographie, ainsi que de toutes les clefs et tous les protocoles nécessaires à son fonctionnement. [**15]**

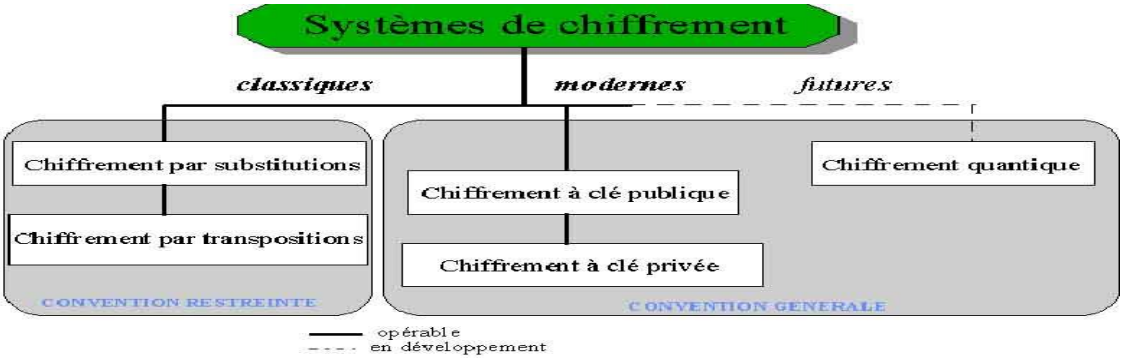

Figure II.1 . Les systèmes de chiffrement

II .2 La cryptographie moderne

À côté de la fonction de chiffrement, qui permet de préserver le secret des données lors d'une transmission, et qui a été utilisée depuis très longtemps, la cryptographie moderne a développé de nouveaux buts à atteindre et qu'on peut énumérer de manière non exhaustive :

- confidentialité, intégrité des données , authentification des divers acteurs

non-répudiation d'un contrat numérique, signature numérique, certification, contrôle d'accès, gestion des clés, preuve de connaissance.

II .3 Principes de la cryptographie moderne :

Dans la société de l'information d'aujourd'hui, l'usage de la cryptographie s'est banalisé. On le retrouve quotidiennement avec les cartes bleues, téléphones portables, Internet ou encore les titres de transport.

La cryptologie moderne a pour l'objet l'étude des méthodes qui permettent d'assurer les services d'intégrité, d'authenticité et de confidentialité dans les systèmes d'information et de communication. Elle recouvre aujourd'hui également l'ensemble des procédés informatiques devant résister à des adversaires.

La cryptologie se partage en deux sous-disciplines :

- La cryptographie qui propose des méthodes pour assurer ces services.
- La cryptanalyse qui recherche des failles dans les mécanismes proposés.

Nous allons détailler dans cet article les grandes lignes des principes de la cryptographie. La principale distinction se fera entre les chiffrements de type symétrique et ceux de type asymétrique.**[16]**

II .2.1 La cryptographie symétrique :

II .2.2.1 Principe général :

Le chiffrement symétrique (aussi appelé chiffrement à clé privée ou à clé secrète) consiste à utiliser la même clé pour le chiffrement et le déchiffrement (Figure II.2). Le chiffrement consiste alors à effectuer une opération entre la clé privée et les données à chiffrer afin de rendre ces dernières inintelligibles. Le déchiffrement consiste à réaliser l'opération inverse c'est-à-dire récupérer le message d'origine à partir du message chiffré en utilisant la clé secrète.

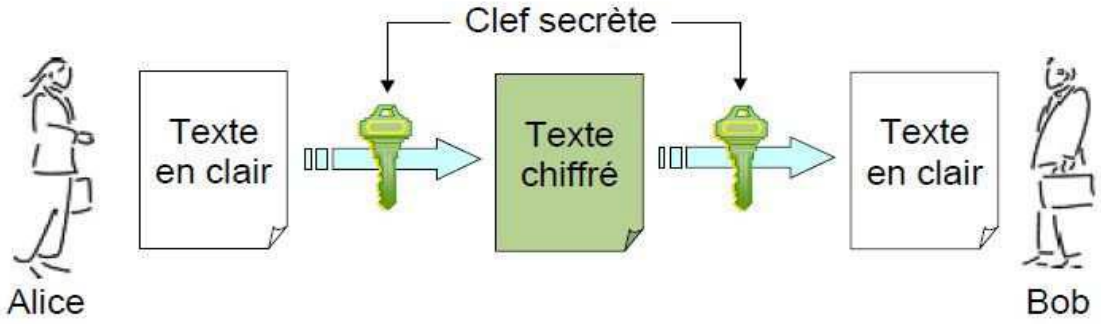

Figure II.2: Chiffrement symétrique

Les algorithmes symétriques sont plus rapides et facilement implémentés sur le matériel. Ils sont des simples opérations de substitution et de transposition et sont mieux adaptés pour une utilisation sur le réseau Internet filaire et en mobilité. Cependant ces systèmes ne sont pas entièrement adéquats pour résoudre tous les problèmes de sécurité. Le processus de chiffrement et déchiffrement dépend de la même clé secrète partagée entre l'émetteur et le récepteur. L'émetteur et le récepteur doivent donc se mettre d'accord sur la clé secrète avant qu'ils puissent communiquer d'une façon sécurisée. Un canal sûr d'échange de clé est difficile à établir. Les crypto systèmes symétriques ont un problème de gestion de clé. En effet lorsqu'un grand nombre de personnes désirent communiquer ensemble, le

nombre de clés augmente de façon importante et la situation devient impraticable. Pour n participants à la communication, on aura besoin de : n(n− 1)/2 clés secrètes enregistrés. Il y a deux catégories de systèmes à clé privée : les chiffrements par blocs et les chiffrements de flux.

II.2.1.2 Chiffrement par bloc (Block Chiper):

C'est une des deux grandes catégories de chiffrements modernes en cryptographie symétrique. Il consiste à un découpage des données en blocs de taille généralement fixe (souvent une puissance de deux comprise entre 32 et 512 bits). Les blocs sont ensuite chiffrés les uns après les autres. Il est possible de transformer un chiffrement de bloc en un chiffrement par flot en utilisant un mode d'opération comme ECB (chaque bloc chiffré indépendamment des autres) ou CFB (on chaîne le chiffrement en effectuant un XOR entre les résultats successifs).

Un exemple de taille de bloc et de clé utilisés par les algorithmes les plus connus:

- DES : blocs de 64 bits, clé de 56 bits.
- IDEA : blocs de 64 bits, clé de 128 bits.
- AES : blocs de 128 bits, clé de 128 à 256 bits.

Pour cette catégorie, nous allons présenter deux algorithmes très connus DES et AES en mettant l'accent sur l'AES car il est devenu le standard recommandé pour le chiffrement symétrique**.[11]**

- **DES (Data Encryptions Standards)** : DES a été développé à la fin des années 1970s comme un standard du gouvernement US pour protéger l'information sensible. Le DES est officiellement défini dans la publication FIPS 46-3 et il est publique. DES est encore supporté dans des outils de cryptographie pour les O.S des dispositifs mobiles et les cartes à puce. C'est un chiffrement qui transforme des blocs de 64 bits avec une clé secrète de 56 bits. Il a une conception basée sur le schéma de Feistel au moyen de permutations et de substitutions. L'attaque de force brute est possible sur une clé DES. Cette attaque a été réalisée par DES Cracker de EFF-Electronic Frontier Fondation en juin 1998 (Elle a trouvé une clé DES dans moins de 3 jours). Avec la technologie actuelle (des CPU très rapides et moins chers), il est hautement possible de cracker DES. La solution a été au

premier temps d'adopter le triple DES (TDES ou 3DES) : 3 applications de DES à la suite avec 2 clés différentes (112 bits) (Figure **II.3**).

Figure II.3 : Algorithme TDES

TDES utilise une taille de blocs de 64 bits et de clé comprise entre 128 bits et 192 bits. TDES est suffisant en sécurité mais il est trois fois plus lent que DES. Un nouvel algorithme remplace DES, c'est l'AES (Advanced Encryption Standard).

- **AES (Advanced Encryptions Standard) :** En 1997, le NIST (National Institute of Standards and Technologie) a lancé un appel d'offre pour un algorithme de chiffrement symétrique avec un bloc de taille 128 bits et supporte des clés de 128 192 et 256 bits. Les critères d'évaluation de l'offre comprenaient la sécurité la puissance de calcul, les contraintes de la mémoire des petits dispositifs (comme la carte à puce), la plateforme logicielle et matérielle et la flexibilité. Un total de 15 algorithmes ont été envoyés et 5 ont été sélectionné parmi ces 15 algorithmes .

  L'algorithme de Rijndael a été sélectionné pour devenir l'AES en 2001. Rijndael a été conçu par Joan Daemen et Vincent Rijmen, deux chercheurs de la Belgique. Les autres algorithmes sont : Serpent, Twofish, RC6, et MARS. L'AES n'utilise pas le schéma de Feistel mais il utilise des opérations mathématiques comme des substitutions, des permutations et des XORs. Il a plusieurs tours identiques (de 10 à 14) et leur nombre dépend de la taille de la clé. L'AES opère au niveau octet ce qui permet une implémentation efficace au niveau matérielle et logicielle. L'AES est un standard, donc libre d'utilisation, sans restriction d'usage ni brevet. NIST spécifie actuellement AES dans le document FIPS 197, comme le nouveau standard de chiffrement symétrique. AES est approuvé pour utilisation par les organisations du gouvernement U.S pour protéger l'information sensible non classifiée.

  AES a trois niveaux forts de sécurité : 128 bits, 192 bits et 256 bits. La sécurité 128 bits fournira au mois 30 ans de protection. AES ne fournit pas seulement une sécurité

supérieure à TDES mais il délivre aussi une meilleure performance. Une meilleure sécurité et une meilleure performance rendent l'AES une alternative plus attractive que TDES et un bon choix pour un algorithme de chiffrement symétrique AES est également un candidat particulièrement approprié pour les dispositifs mobiles limités en ressources de calcul et de stockage. Le monde de la 3G (3ème génération de dispositifs mobiles) a adopté l'algorithme AES pour son schéma d'authentification.

Autres finalistes d'AES , Durant la compétition avec AES, Serpent et Twofish avaient une performance faible comparée à Rijndael. RC6 et MARS ont des problèmes de sécurité et d'efficacité. RC6 a été cassé à au moins de 17 rounds sur 20 pendant la compétition avec AES. MARS était plus couteux en calcul pour l'implémenter comparé aux autres finalistes d'AES.**[17]**

II.2.1.3 Chiffrements de flux (Stream Cipher):

Les algorithmes de chiffrement de flux peuvent être définis comme étant des algorithmes de chiffrement par blocs, où le bloc a une dimension unitaire (1 bit, 1 octet, etc.) ou relativement petite. Leurs avantages principaux viennent du fait que la transformation (méthode de chiffrement) peut être changée à chaque symbole du texte clair et du fait qu'ils soient extrêmement rapides. De plus, ils sont utiles dans un environnement où les erreurs sont fréquentes car ils ont l'avantage de ne pas propager les erreurs (diffusion). Ils sont aussi utilisés lorsque l'information ne peut être traitée qu'avec de petites quantités de symboles à la fois.

Quelques algorithmes de cryptographie symétrique par flot:

- A5 : utilisé dans les téléphones mobiles de type GSM pour chiffrer la communication par radio entre le mobile et l'antenne-relais la plus proche.
- RC4, le plus répandu, conçu par Ronald Rivest, utilisé notamment par le protocole WEP, un algorithme récent de Eli Biham – E0 utilisé par le protocole Bluetooth.

II.2.1.4 Considérations de sécurité pour la cryptographie symétrique :

Lorsqu'on utilise la cryptographie à clé symétrique, nous devons considérer certains principes qui constituent les critères de base pour implémenter une sécurité fiable et efficace.

II.2.1.5 Gestion de clé :

Il est fondamental d'utiliser un générateur fort de nombre pseudo aléatoire (PRNG – Pseudo Random Number Generator) pour générer des sorties d'octets aléatoires utilisées

comme clés. Pour l'échange de la clé, on peut utiliser un centre de distribution de clé ou une cryptographie à clé asymétrique pour l'échange de la clé secrète. Généralement les applications utilisent un algorithme symétrique pour chiffrer les données, et un algorithme asymétrique pour chiffrer la clé secrète. Cette méthode mixte symétrique et asymétrique est mieux sécurisée et plus efficace. Si la clé est utilisée pour une communication entre deux parties sur un réseau, il est recommandé d'utiliser une clé de session parce qu'elle limite la chance à un attaquant de trouver la clé exacte. Si une clé est exigée pour chiffrer des fichiers locaux sur une machine, alors les clés générées par mot de passe (PBE-Password Based Encryption) peuvent être utilisées, mais il faut s'assurer que le mot de passe lui-même ne soit pas compromis. Donc une règle fondamentale de la cryptographie symétrique : Utiliser des clés de session pour chiffrer une communication réseau et des clés PBE (basés sur des mots de passe) pour protéger les données sensibles sur la machines.

II.2.1.6 Authentifier le chiffrement symétrique :

Les modes de chiffrement protègent les données de l'écoute mais ils ne fournissent aucune authentification. La fonction de déchiffrement déchiffre juste les données, elle peut déchiffrer un texte chiffré déjà modifié, en un certain plaintext qui est différent du plaintext d'origine.

Dans un tel cas, il est recommandé de produire un code d'authentification du message MAC (Message Authentiquassion Code), du texte en clair et l'ajouter au texte chiffré. Utiliser le chiffrement en mode bloc au lieu du chiffrement par flot : Il est recommandé d'utiliser le mode de chiffrement en bloc (CBC ou CTR) avec un vecteur d'initialisation générée d'une manière aléatoire. Un vecteur d'initialisation mal choisi peut causer une faiblesse de sécurité (le cas de RC4 avec le protocole WEP).

II.2.2 Cryptographie asymétrique :

La cryptographie asymétrique se base sur des problèmes mathématiques complexes (factorisation de grands nombres entiers ou équation de logarithme discrète). La cryptographie asymétrique se base sur le principe de deux clés: clé publique, clé privée (Figure **II.4**). La clé publique est mise à la disposition de quiconque désire chiffrer un message (cette clé peut être connu par tout le monde). Ce dernier ne pourra être déchiffré qu'avec la clé privée, qui doit être confidentielle et connue seulement par son propriétaire.

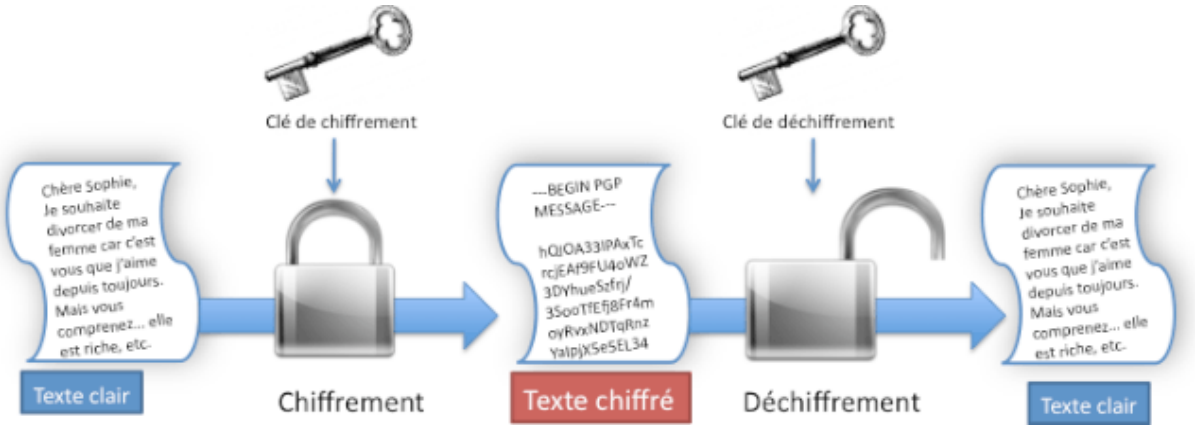

Figure II.4 : Chiffrement Asymétrique

Un chiffrement asymétrique est défini par trois algorithmes

- Algorithme de génération des clés
- Algorithme de chiffrement
- Algorithme de déchiffrement ;

Certains algorithmes asymétriques comme RSA offre aussi des opérations pour la génération de signature numérique et sa vérification.

L'utilisation d'une paire de clés publique/privée permet d'assurer la confidentialité, l'authentification, l'intégrité et l'échange de la clé secrète. Cependant les algorithmes asymétriques ne réalisent pas tous ces fonctions.

La Table **II**.1 donne un résumé des opérations cryptographiques pouvant être réalisés par les algorithmes asymétriques les plus connus.

Table II.1 : Les applications des crypto systèmes asymétriques

Algorithme	Chiffrement/Déchiffrement	Signature numérique	Echange de clé
RSA	Oui	Oui	Oui
Diffie-Hellman	Non	Non	Oui
DSA	Non	Oui	Non
EC-Elliptic Curves	Oui	Oui	Oui

Cryptographie à clef publique basé sur des problèmes mathématiques difficiles à résoudre :

II.2.2.1 Problème de la factorisation : Le problème de la factorisation consiste à retrouver la décomposition en facteurs premiers d'un entier obtenu de manière secrète par multiplication de deux nombres premiers de taille comparable.
Un tel nombre composé est classiquement appelé « module ». L'utilisation du problème de la factorisation est en général effectuée au moyen du crypto systèmes RSA**.[12]**

II.2.2.2 Cryptage RSA :

Le cryptage RSA, du nom de ses concepteurs, Ron Rivest, Adi Shamir et Leonard Adleman, est le premier algorithme de chiffrement asymétrique. Il a été découvert en 1977 au Massachusetts Institute of Technologie. Le principe de ce cryptage est d'utiliser une clé publique pour crypter les données et une clé privée qui servira à les décrypter. Nous allons maintenant décrire les opérations qu'on peut réaliser avec cet algorithme :

Génération des clés :

Le RSA fonctionne à partir de deux nombres premiers. Ces deux nombres doivent être très grands et calculer la clé privée et la clé publique comme dans l'algorithme suivant:

Algorithme de Génération des clés :

Entrées : Deux nombres premiers p et q .

Sorties : Une clé privée d et une clé publique e.

€Si p = q alors Prendre un nombre aléatoire p et q.

Sinon N = p q.

Fin Si

Calcule $\phi n = (p - 1)(q - 1)$.

Prendre un nombre aléatoire e dans l'intervalle $[1, \phi n]$.

Si **pgcd** $(e , \phi n) \neq 1$ alors

Prendre un nombre aléatoire e dans l'intervalle $[1, \phi n]$.

Sinon Calcule $d \equiv e^{-1} \bmod (\phi n)$.

Fin Si

Le chiffrement d'un message :

Supposons maintenant que les intervenants A et B possèdent chacun son module RSA et ses clés N_A, e_A , d_A pour A et N_B, e_B, d_B pour B. Si A veut envoyer un message M à B, il peut procéder comme dans l'algorithme suivant:

Algorithme de Chiffrement d'un message :

Entrées : Un message clair M et la clé publique (N_B, e_B).

Sorties : Un message chiffré C.

Transformer le message en un nombre entier M de l'intervalle $[0, N_B-1]$

Calculer $C \equiv M^{e_B} \bmod N_B$ et $C < N_B$.

Envoyer le message C.

Déchiffrement du message :

B a reçu un message chiffré C de la part de A. Alors B peut le déchiffrer en utilisant sa clé secrète $d+$ comme dans l'algorithme suivant :

Algorithme de Déchiffrement d'un message :

Entrées : Un message chiffré C et la clé privée (N_B, d_B).

Sorties : Un message clair M.

Calculer $M \equiv C^{d_B} \bmod N_B$.

Retourner le message M .

Signature d'un message :

Supposons que A veut envoyer un message M à B. Comment B peut-il être sûr que le message reçu a bien été envoyé par A. Pour résoudre ce problème, RSA peut être utilisé aussi bien pour transmettre un message que pour le signer. Tout d'abord, A clé utilise la clé publique (N_B, e_B) de B pour transmettre le message chiffré C et produire une signature S du message à l'aide de sa propre clé privée (N_A, d_A) **.**

Algorithme de Signature d'un message :

Entrées : Un message clair M et clé privée (N_A, d_A) , fonction de hachage h .

Sorties : Un message chiffré C et sa signature S.

A transforme le message en un nombre entier M de l'intervalle [0 N_B-1].

Calculer $D \equiv h(M)$;

Calculer $S \equiv D^{d_A} \bmod N_A$;

Retourner le message C et la signature S.

Vérification d'une signature :

Pour vérifier la signature, nous utilisons alors l'algorithme suivant : chiffré C de la part de A. Alors B peut le déchiffrer en utilisant la clé publique du signataire comme dans l'algorithme suivant :

Algorithme de Vérification d'une signature :

Entrées :Un message chiffre C, une signature S et la clé publique (N_A ,e_A),et la clé privée (N_B, e_B)

Sorties : Verification d' un signature.

B déchiffre le message C : Calculer $C \equiv M^{d_B} \bmod N_B$

B Calcule $D' \equiv S^{e_A} \bmod N_A$ et D = h(M)

Si D' = D Alors la signature est vérifiée .

II.2.2.3 Considérations de sécurité pour l'utilisation de RSA :

- Il est fondamental d'utiliser CRT (Chinese Remainder Theorem) dans RSA pour augmenter la vitesse de déchiffrement.
- Il est aussi fondamental d'utiliser le schéma de cryptage RSAES-OAEP au lieu de RSA seul, pour les opérations de chiffrement et déchiffrement.
- Il est fondamental d'utiliser une clé RSA 2048-bit comme un minimum pour l'utilisation de la cryptographie RSA dans les nouvelles applications (à partir de 2010).

II.2.2.4 Problème Logarithme discret :

Le premier système basé sur le problème de logarithme discret (DLP) était le protocole d'échange de clés Diffie et Hellman en 1976. En 1984, ElGamal décrit un crypto système basé DLP pour le chiffrement à clé publique et schémas de signature. Nous allons maintenant décrire cet algorithme qui permet :

Génération de paramètres de DL

Algorithme de génération de paramètres de DL :

Entrées: paramètres t, l.

Sorties: paramètres (p, q, g)

1. Sélectionner t un nombre premier q de t-bit et un nombre premier p de l-bit tel que q divise $p-1$
2. choisir un élément g en ordre q.
3. Sélectionnez arbitraire $h \in [1, p-1]$ et Calcule $g = h^{(p-1)/q} \bmod p$.
4. Si $g = 1$ alors aller à la phase 3.
5. Retourner (p, q, g)

Génération des clés DL:

Algorithme de génération des clés DL :

Entrées: paramètres (p, q, g) .

Sorties: clé publique y , clé privée x.

1. Sélectionner $x \in R\ [1\ ,\ q\ -1]$.
2. Calcule $y = g^x \bmod p$.
3. Retourner (y, x).

II.2.2.5 Autres algorithmes Asymétrique :

Protocole d'échange de clés DH (Diffie-Hellman) : algorithme à clé publique d'échange de clé, basé sur le problème de logarithme discret, développé par Diffie et Hellman en 1976.ce

protocole permet à deux tiers de générer un secret partagé sans avoir aucune information préalable l'un sur l'autre et en échangeant uniquement leurs clés publiques.

Voici le déroulement de l'algorithme :

1. Alice choisit de manière aléatoire un grand nombre entier a, qu'elle garde secret et calcule sa valeur publique, $A = g^{a} \bmod n$. Bob fait de même et génère b et $B = g^{b} \bmod n$.
2. Alice envoie A à Bob, Bob envoie B à Alice ;
3. Alice calcule $K_{ab} = B^{a} \bmod n$, Bob calcule $K_{ba} = A^{b} \bmod n$, $K_{ab} = K_{ba}\ g^{ab} \bmod n$ c'est le secret partagé par Alice et Bob.

DSA (Digital Signature Algorithm): Diffie-Hellman permet de crée un secret commun (et donc de chiffrer des communications) mais contrairement à RSA, il ne permet pas de signer des documents. C'est pour cette raison que Diffie-Hellman est souvent associé à DSS (Digital Signature Standard, un autre algorithme) ou DSA. DSS permet de signer les documents.

DSA est une norme promulguée par le NIST. Il est uniquement utilisé pour les signatures numériques. DSA génère des signatures plus rapides, et peut vérifier les signatures RSA.

Nous allons maintenant décrire l'algorithme DSA qui permet :

Génération de signature de DSA :

Algorithme de DSA signature génération :

Entrées: DL paramètres (p, q, g), clé prive x , message M.

Sorties: Signature (r,s).

1. Choisir un nombre entier k aléatoirement dans l'intervalle $]1, q-1[$.
2. Calcule $T = g^{k} \bmod p$.
3. Calcule $r = T \bmod q$. Si $r = 0$ alors aller à phase 1.
4. Calcule $h = H(M)$.
5. Calcule $s = K^{-1}(h + xr) \bmod q$. Si $s = 0$ alors aller à phase 1.
6. Return (r, s).

Vérification de signature de DSA :

Algorithme de DSA signature vérification :

Entrées: DL paramètres (p, q, g), clé publique y, message M, signature (r, s).

Sorties: Vérification d'une signature.

1. Vérifier que (r, s) soient dans l'intervalle [1, q-1].

2. Si n'est pas vérifier Alors return ("Signature rejetée ").

3. Calcule h = H (M).

4. Calcule $w = s^{-1} \bmod q$

5. Calcule u1 = hw mod q et u2 rw mod q .

6. Calculer T = gu1 + yu2 mod p

7. Calculer r = T mod q

8. Si r = r' alors Retourner ("Signature acceptée ")

Sinon Retourner ("Signature rejetée ").

II.2.2.6 Problème du Logarithme Discret pour les courbes elliptiques:

Il est également possible de définir des problèmes de logarithme discret dans des structures plus complexes pour lesquels aucun algorithme plus efficace que les méthodes génériques de calcul de logarithme discret n'est connu. C'est en particulier aujourd'hui le cas des courbes elliptiques qui sont définies
sur un corps de base pouvant être, en pratique, premier GF(p) ou binaire $GF(2^n)$.

II.2.3 les avantages et les Inconvénients de la cryptographie symétrique et asymétrique :

Cryptographies	Advantages	Inconvénients
Symétrique	-Système rapide de chiffrement / déchiffrement . -Clés relativement courtes (128 ou 256 bits) -Bonnes performances et sécurité bien étudié.	-Gestion des clés difficiles (nombreux clés) -Point faible = l'échange de la clé secrète . -Dans un réseau de N entités susceptibles de communiquer secrètement il faut distribuer N*(N-1)/2 clés.
Asymétrique	-Gestion de la secrète facilitée -Pas de secrète à transmettre . -Nombre de clés à distribuer est réduit par rapport aux clés symétriques . -Très utile pour échanger les clés . -Permet de signer des messages Facilement	-La relation clé publique /clé privée impose : -Des clés plus longues (1024 à 4096 bits). -Gestion de certificats de clés publiques . -Lenteur de calcul . -Pas d'authentification de la source .

Table II.2 : les avantages et les Inconvénients symétrique / asymétriques

II.2.4 Les systèmes cryptographiques hybrides ou mixtes:

La cryptographie hybride est un système de cryptographie faisant appel aux deux grandes familles de systèmes cryptographiques : la cryptographie asymétrique et la cryptographie symétrique. Les logiciels comme PGP et GnuPG reposent sur ce concept qui permet de combiner les avantages des deux systèmes.

Principe :

La cryptographie asymétrique est intrinsèquement lente à cause des calculs complexes qui y sont associés, alors que la cryptographie symétrique brille par sa rapidité. Toutefois, cette dernière souffre d'une grave lacune, on doit transmettre les clés de manière sécurisée (sur un canal authentifié). Pour pallier ce défaut, on recourt à la cryptographie asymétrique qui travaille avec une paire de clés : la clé privée et la clé publique. La cryptographie hybride

combine les deux systèmes afin de bénéficier des avantages (rapidité de la cryptographie symétrique pour le contenu du message) et utilisation de la cryptographie "lente" uniquement pour la clé.

Chiffrement :

La plupart des systèmes hybrides procèdent de la manière suivante. Une clé aléatoire, appelée clé de session, est générée pour l'algorithme symétrique (3DES, IDEA, AES et bien d'autres encore), cette clé fait généralement entre 128 et 512 bits selon les algorithmes. L'algorithme de chiffrement symétrique est ensuite utilisé pour chiffrer le message. Dans le cas d'un chiffrement par blocs, on doit utiliser un mode d'opération comme CBC, cela permet de chiffrer un message de taille supérieure à celle d'un bloc.

La clé de session quant à elle, se voit chiffrée grâce à la clé publique du destinataire, c'est ici qu'intervient la cryptographie asymétrique (RSA ou ElGamal). Comme la clé est courte, ce chiffrement prend peu de temps. Chiffrer l'ensemble du message avec un algorithme asymétrique serait bien plus coûteux, c'est pourquoi on préfère passer par un algorithme symétrique. Il suffit ensuite d'envoyer le message chiffré avec l'algorithme symétrique et accompagné de la clé chiffrée correspondante. Le destinataire déchiffre la clé symétrique avec sa clé privée et via un déchiffrement symétrique, retrouve le message. **[18]**

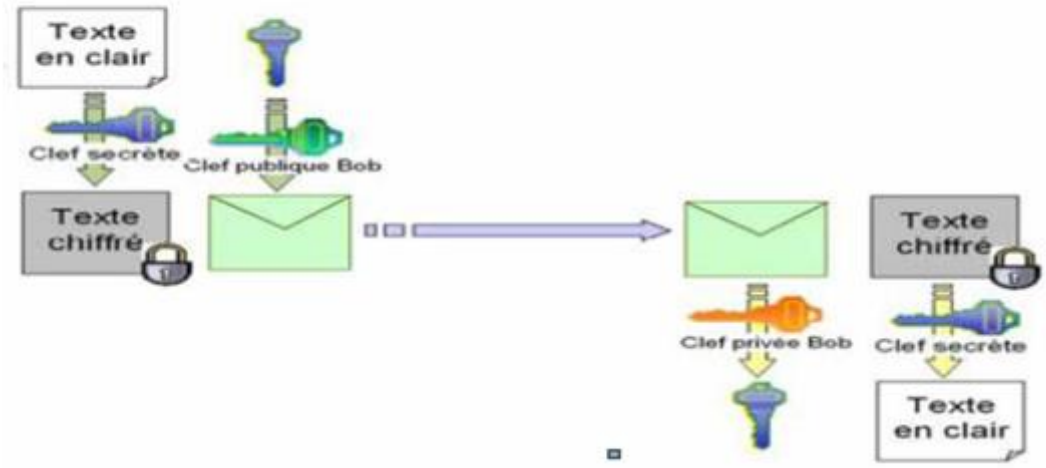

Figure II.5 : Chiffrement hybrides

II.5 Fonctions de hachage :

Une fonction de hachage est aussi appelée fonction de hachage à sens unique ou "one-way hash function" en anglais est une fonction qui associe à un grand ensemble de données un ensemble beaucoup plus petit .Le résultat de cette fonction est par ailleurs aussi appelé somme de contrôle, empreinte, résumé de message, condensé ou encore empreinte cryptographique lorsque l'on utilise une fonction de hachage cryptographique. Les fonctions de HASH peuvent être utilisées pour l'authentification de message, dérivation de clé, génération de nombre pseudo aléatoire et génération de la signature numérique.

Une fonction de hachage H doit avoir les propriétés suivantes :

1. H peut être appliqué à un bloc de données de toute taille.
2. H produit une sortie de longueur fixe.
3. H(x) est relativement facile à calculer pour toute donnée x.
4. Pour toute empreinte (digest) d, il est informatiquement impossible de trouver x tel que H(x) = d .
5. Pour tout bloc de données x, il est informatiquement impossible de trouver $y \neq x$ avec H(y) = H(x)
6. Il est informatiquement impossible de trouver toute paire de messages (x, y) tel que H(x) = H(y).

Une fonction HASH qui satisfait les 5 premiers points seulement est considérée comme faible (weak hash function), et si la sixième propriété est satisfaite, elle est alors une fonction HASH forte. Nous nous concentrons sur l'algorithme SHA (Secure Hash Algorithm) parce qu'il est le plus robuste et le plus utilisé.

II.4.1 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**.[19]**
- La liste des algorithmes SHA est : SHA-1, SHA-256, SHA-384 et SHA-512. A cause des calculs intensifs dans SHA-384 et SHA-512, ces deux algorithmes ne sont pas recommandés pour être utilisés sur les dispositifs mobiles SHA-1 est l'algorithme le plus utilisé dans la famille des algorithmes SHA. SHA-1

prend en entrée un message et produit une empreinte de 160 bits. SHA-256 prend aussi en entrée un message mais produit une empreinte de 256 bits.

- Il y a une nouvelle attaque sur SHA-1 sans utiliser la force brute, qui trouve des collisions à l'intérieur de 269 opérations hash.

- A cause de l'avancement technologique et les attaques, NIST a planifié d'abandonner SHA-1 la fin 2010, et recommande des fonctions HASH plus fortes comme SHA-256.
- Les algorithmes SHA-256, SHA-384 et SHA-512 sont relativement nouveaux et sont aussi standardisés par NIST mais ils sont plus intensifs en calcul et crypto analyses**.[2]**

II.4.2 Autres algorithmes de hachage :

Il y a d'autres algorithmes de hachage comme MD4, MD5 et SHA-0, mais ils sont plus exposés aux attaques de collision. Il est donc recommandé d'utiliser SHA-256 comme fonction de hachage**.[10]**

II.5 Code d'authentification de message MAC :

MAC-Message Authentification Code partage les 5 premières propriétés des fonctions de hachage à l'exception qu'il implique l'utilisation d'une clé secrète comme une entrée avec le message à authentifier. Avec MAC nous avons la fonction : M(s, x) = d avec M est la fonction MAC qui génère d à partir d'un secret s et un message x. Une fonction de hachage ne peut pas être utilisée comme MAC parce qu'elle n'exige pas une clé secrète.

- Une fonction HASH peut opérer d'une manière identique à un MAC comme H (s||x) , H (s||x)ou H (s||x ||s) . Il est donc fondamental d'éviter l'utilisation de fonction de hachage comme un MAC**.[3]**

- Il y avait plusieurs approches pour incorporer une clé secrète dans un algorithme de HASH existant. L'approche la plus connue est HMAC (keyed-Hash Message Authentification Code). HMAC a été choisi pour implémenter MAC pour le protocole IPSec. Il est aussi utilisé dans d'autres protocoles comme TLS et SET (Secure Electronic Transaction).

II.5.1 HMAC :

Les objectifs pour l'implémentation de HMAC :

- Utiliser sans modification des fonctions de hachage disponibles.

- Permettre de remplacer la fonction HASH intégrée dans le cas ou des fonctions HASH plus rapides, plus sécurisées sont trouvées ou exigées.
- Préserver la performance originale de la fonction HASH sans ajouter une dégradation significative.
- Utiliser et gérer des clés d'une manière simple.

HMAC est comme une boite noire qui accepte une fonction hash à intégrer comme une partie de son implémentation. L'important, si la fonction hash est craquée, elle pourra être remplacée par une autre plus sécurisée. Le HMAC sera plus sécurisée si la fonction hash intégrée est sécurisée.

HMAC fournit aussi un support pour éviter les problèmes d'extension de longueur et les collisions de message partiel. La valeur de la clé secrète utilisée comme une entrée dans MAC est aussi protégée contre les attaques de recouvrement de clé.

II.5.2 Autres fonctions MAC :

CBC-MAC est une méthode pour transformer un chiffrement de bloc en MAC. CBC-MAC est considérée sécurisée si le chiffrement utilisé est sécurisé. Cependant il est dangereux d'utiliser la même clé secrète pour le chiffrement CBC et l'authentification CBC-MAC.

UMAC est une fonction plus rapide que HMAC mais elle est limitée par une taille faible de l'empreinte (64 bits) et la complexité de ces différentes implémentations.

Donc HMAC est plus facile à utiliser et plus robuste comparé aux autres algorithmes MAC. Il est recommandé d'utiliser HMAC comme fonction MAC.

II.6 Certificats à clé publique et les signatures numériques :

Le format de certificat le plus largement utilisé est la version 3 du certificat X.509. Le standard ISO pour une structure de certificat numérique est basé sur le format X.509. Le format X.509 définit un Framework pour fournir des services d'authentification à travers le répertoire X.500 qui consiste à un dépôt de certificats à clé publique. Chaque certificat consiste à la clé publique d'un utilisateur et elle est signée par la clé privée d'une CA. X.509 a été initialement délivré en 1988, des recommandations de révisions ont été diffusées en 1993 pour adresser certains problèmes de sécurité avec les versions 1 et 2. La version 3 est sortie en

1995. X.509 inclut aussi des standards pour la liste de révocation de certificat (CRL). X.509 est actuellement utilisé dans les protocoles de sécurité comme SSL/TLS, IPSec, S/MIME, et SET.

II.6.1 Certificats à clé publique et le format X.509 :

Les certificats à clé publique X.509 sont crées par une CA de confiance et placés dans un répertoire par la CA ou par l'utilisateur. Le serveur de répertoire fournit un emplacement accessible aux utilisateurs pour obtenir les certificats. Les certificats sont codés en utilisant la syntaxe ASN.1 (Abstract Syntaxe Notation 1) de l'OSI. Les certificats utilisateur générés par la CA ont les caractéristiques suivantes : **[4]**

- Tout utilisateur avec l'accès à la clé publique de la CA peut récupérer la clé publique de l'utilisateur qui a été certifié.
- Aucune partie autre que la CA ne peut modifier le certificat sans qu'elle soit détectée. Les champs du certificat X.509 **[4, 5]**
- Version : Indique à quelle version de X.509 correspond ce certificat.
- Sérial number : Numéro de série du certificat (propre à chaque autorité de certification) .
- Signature Algorithm ID : Identifiant du type de signature utilisée .
- Issuer Name : l'entité qui a signé et délivré le certificat
- Validityperiod : comprend deux dates : la date de début de validité et la date de la fin de validité
- Subject Name : identifie l'entité associée avec la clé publique.
- Subject public key info : information sur la clé publique de ce certificat.
- Issuer Unique ID/Subject Unique ID : Extensions optionnelles introduites avec la version 2 de X.509.

Extensions : les extensions définies pour les certificats X.509 v3 fournissant des. Signature : la valeur du hash de tous les champs du certificat chiffrée par la clé privée de la CA .

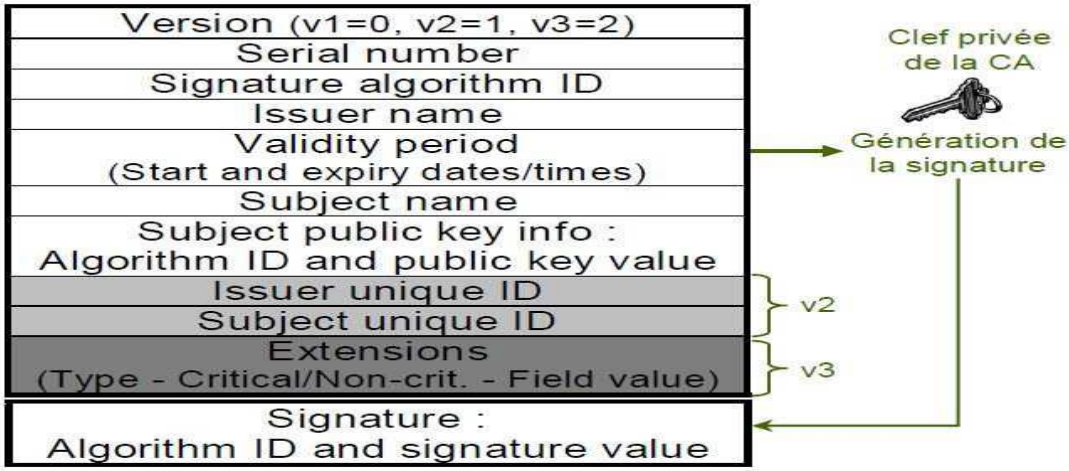

Figure II.6 : Certificat X.509

II.6.2 Signatures numériques :

Les signatures numériques associées à des certificats à clé publique fournissent l'authentification, l'intégrité et la non-répudiation. Il ya plusieurs schémas de signatures standardisés. Le schéma PKCS#1 v2.1 supporte le cryptage RSA et il a deux schémas de signatures basés RSA : RSASSA-PSS et RSASSA-PKCS1-v1_5. Bien qu'il n'y ait pas d'attaques connues sur le schéma de signature RSASSA-PKCS1-v1_5, le standard PKCS#1 v2.1 recommande RSASSA-PSS pour les nouvelles applications à cause de sa robustesse. RSASSA-PKCS1-v1_5 est inclut dans PKCS#1 v2.1 pour compatibilité avec les applications existantes. Malheureusement, le schéma de signature PSS a un brevet appliqué par l'université de la Californie, mais si PPS est inclut dans un standard IEEE et si ce standard sera adopté, une licence libre est offerte pour utiliser PSS comme schéma de signature numérique . Le schéma de signature PSS est spécifié dans IEEE P1363lorsque PKCS#1 v2.1 a été écrit. Il est recommandé d'utiliser un schéma de signature qui n'est pas protégé par un des brevets. Le schéma de signature ECDSA qui est une variante de DSA mais en utilisant les courbes elliptiques, est plus efficace que celui du RSA Il a des tailles de clés plus petites par conséquent la signature numérique produite occupe moins d'octets que celle produite par RSA.

Il est fondamental d'utiliser un algorithme de signature facile, efficace et sécurisé comme RSA ou ECDSA. Le schéma de signature ECDSA est préféré mais il n'est pas encore largement adopté**.[6]**

Chapitre III :

LA cryptographie dans java et la package cryptographique de bouncy castle

III.1 Introduction :

La plateforme Java a été conçue sur une base de sécurité. Le langage Java lui-même est de type sécurisé. La plateforme a évoluée et a élargi son intervalle de déploiement, l'architecture de la sécurité java a aussi évolué pour supporter un ensemble de services cryptographiques.

Actuellement, l'architecture comprend un ensemble très large d'APIs, d'outils, d'implémentations d'algorithmes, et de protocoles de sécurité. Java fournit au développeur un Framework de sécurité pour écrire des applications, et aussi à l'utilisateur ou l'administrateur un ensemble d'outils pour la gestion sécurisée des applications.

Les APIs de sécurité Java couvrent plusieurs secteurs. Les interfaces de cryptographie et de PKI (Public Key Infrastructure). Ces interfaces vérifiant l'authentification et le contrôle d'accès, permettent aux applications de se protéger contre les accès non autorisés aux ressources **[17].** Pour les services de cryptographie, nous présentons l'architecture de cryptographie JCA et son extension JCE. Nous montrons les principes de conception de JCA/JCE, la notion de CSP (Cryptographic Service Provider). Nous présentons les concepts introduits dans l'API de sécurité comme les classes de moteur (engine classes) et l'interface de provider de service SPI (Service Provider Interface) pour l'implémentation des providers de service. Dans la classe Provider, nous montrons comment installer et configurer un provider et comment utiliser ses implémentations ? Nous prenons comme cas pratique le provider JCE de Bouncy Castle, nous montrons son installation et sa configuration dans la plateforme J2SE**.[7]**

III.2 Sécurité du langage Java :

Le langage Java est conçu pour être de type sécurisé et facile à utiliser. Le langage définit plusieurs modes d'accès qui peuvent être affectés aux classes, aux méthodes et aux variables Java. Il définit quatre niveaux d'accès différents: private, protected, public, et package. Le mode d'accès le plus utilisé est public pour une utilisation ouverte et sans restriction. Le mode le plus restrictif est private qui n'est pas accessible en dehors de la classe. Le mode protected permet l'accès à toute classe ou sous classe à l'intérieur du même package. Le niveau d'accès package permet seulement l'accès aux classes à l'intérieur du même package **[8].**

Le compilateur Java produit un programme Java au format byte-code qui s'exécute sur une machine virtuelle Java indépendamment de la machine et de la plateforme OS. Un vérificateur du byte-code est utilisé automatiquement pour s'assurer que le programme exécuté dans l'environnement d'exécution Java (JRE) est en byte-code. Une fois le byte-code est vérifié, l'environnement d'exécution Java le prépare pour l'exécution **[7]**

III.3 Cryptographie dans Java :

III.3.1 Concepts cryptographiques implémentés par l'API de sécurité Java:

La spécification de l'API de sécurité Java utilise des termes techniques pour désigner des concepts cryptographiques. On donne la signification exacte de ces termes utilisés **[8]:**

- Chiffrement et déchiffrement (Encryption and Decryption): Le chiffrement est le processus qui prend des données en entrée (plaintetxt) et une clé (key) pour produire des données chiffrées (ciphertext). Le déchiffrement est le processus inverse qui prend le texte chiffré et une clé pour reproduire les données originales en texte clair.
- Chiffrement basé mot de passé (PBE-Password-Based Encryption) : Le chiffrement basé mot de passe (PBE) permet de créer une clé de chiffrement à partir d'un mot de passe. Pour éviter les attaques de mot de passe (ou attaque de dictionnaire).
- Agrément de clé (Key Agreement) ou établissement de clé : C'est un protocole qui permet d'établir des clés cryptographiques entre deux ou plusieurs parties.
- Code d'authentification de message (MAC-Message Authentification Code) : Un code d'authentification de message (MAC) fournit un moyen pour vérifier l'intégrité de l'information transmise ou stockée dans un endroit incertain, basé sur une clé secrète. Le mécanisme MAC qui est basé sur une fonction de hachage et un secret, est désigné sous le nom *HMAC*. Un *HMAC* peut être utilisé avec toute fonction de hachage, e.g. MD5 ou SHA1, en combinaison avec une clé secrète partagée .

III.3.2 Services cryptographiques fournis :

L'architecture de la cryptographie Java est un Framework pour l'accès et le développement de fonctionna0lités cryptographiques. Elle comprend des APIs pour un grand nombre de services cryptographiques, comprenant**[1]:**

Algorithmes d'empreinte de message

- Algorithmes de signature numérique
- Chiffrement en mode bloc (block cipher)
- Chiffrement symétrique en mode flux de données (streamcipher)
- Chiffrement basé mot de passe (PBE)
- Cryptographie de courbe elliptique (ECC)
- Algorithmes d'agrément de clé
- Générateurs de clés secrètes et de paire de clés
- Codes d'authentification de message (MACs)
- Générateurs de nombres pseudo aléatoires (PRNG)

A cause des contrôles d'exportation imposés par le gouvernement U.S sur les logiciels de cryptographie, les APIs Java cryptographiques sont organisés dans deux packages différents :

- Le package java.security contient des classes non concernées par les contrôles
- Le package javax.crypto contient des classes concernées par les contrôles d'exportation (comme Cipher et KeyAgreement).

III.4 Architecture JCA/JCE :

III.4.1 Présentation des architectures JCA et JCE :

L'API de sécurité fait partie du noyau du Java. Elle est conçue pour permettre aux développeurs d'incorporer des fonctionnalités de sécurité dans leurs programmes .

III.4.1.1 JCA (Java Cryptographie Architecture):

L'API de sécurité dans JDK 1.1 a introduit l'architecture de la cryptographie Java-JCA (Java Cryptography Architecture), qui est un Framework pour l'accès et le développement de fonctionnalités cryptographiques pour la plateforme Java. JCA comprenait des APIs pour signatures numériques et empreintes de messages. C'est une architecture basée provider permettant d'implémenter un ou plusieurs providers de services cryptographiques interopérables, connus comme providers de services cryptographiques- CSPs (Cryptographic Service Providers). L'API JCA s'appuie sur le package *java.security* **[9].**

III.4.1.2 JCE (Java Cryptographie Extension) :

JCE est une API qui propose de standardiser l'utilisation de la cryptographie en restant indépendant des algorithmes utilisés. Elle prend en compte le chiffrement/déchiffrement de données, la génération de clés et l'utilisation de la technique MAC (Message Authentification Code) pour garantir l'intégrité d'un message.

JCE a été intégrée au JDK 1.4. Pour pouvoir utiliser cette API, il faut obligatoirement utiliser une implémentation développée par un fournisseur (provider). Avec le JDK 1.4, Sun fournit une implémentation de référence nommée SunJCE. Les classes et interfaces de l'API JCE sont regroupées dans le package

JCA/JCE sont architecturés pour fournir une couche d'abstraction pour les développeurs d'application, les objets fournis par l'implémentation (les algorithmes) sont crées en utilisant des classes de production (factory classes).

Cette architecture est basée provider, ce qui signifie que JCA/JCE fournissent une implémentation d'un ensemble d au développeur. JCA/JCE ont des mécanismes simples pour permettre aux développeurs d'ajouter et de choisir des providers particuliers. La Figure III.1 montre comment les différents composants

de l'application fait appel aux classes appropriées des APIs JCA/JCE, ces dernières appellent les classes dans un provider qui fournit des implémentations pour les classes SPI. Les classes utilisent le code interne dans le provider pour fournir la fonctionnalité demandée .

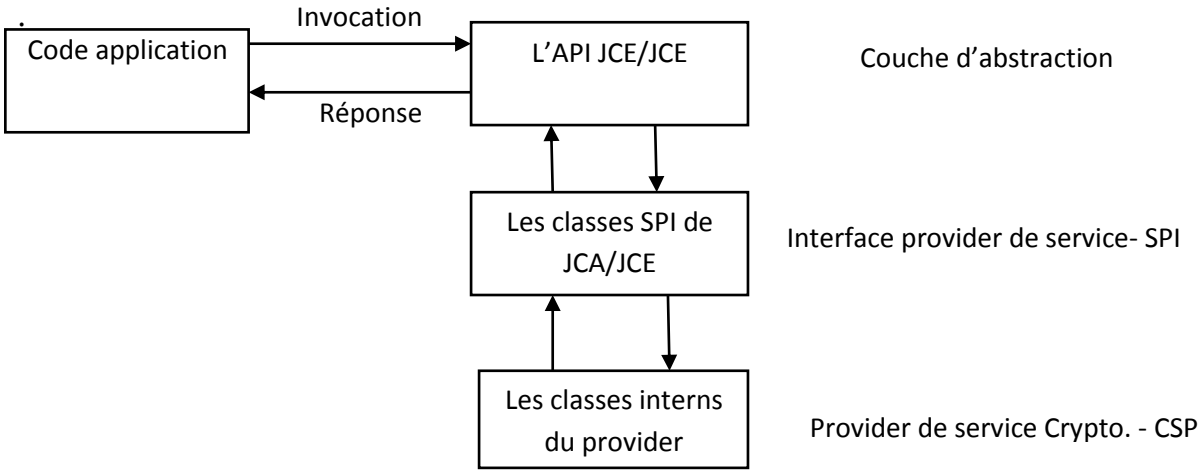

Figure III.1 : L'API JCA/JCE et les providers de service

Sun a introduit l'utilisation des fichiers de restrictions « policy files » qui permettent l'utilisation des algorithmes et les tailles de clés avec restriction, il a aussi introduit l'idée de

Providers signés. Les changements sont signés par l'agence nati (National Security Agency) et le département de commerce U.S.

III.4.2 Principes de conception des providers dans JCA/JCE :

JCA/JCE a deux principes de conception de providers :

- Leur indépendance de l'implément services cryptographiques sans connaitre les détails de l'implémentation .
- Leur indépendance des algorithmes et leur extensibilité : des nouveaux providers de services et/ou des algorithmes peuvent être ajoutés sans affecter les providers existants .

Ces deux principes fournissent une architecture modulaire ce qui permet de réaliser un chiffrement par une implémentation et le déchiffrement par une autre implémentation .

L'indépendance de l'implémentation et l'indépendance de l'algorithme sont complémentaires, on peut utiliser des services cryptographiques comme les signatures numériques et les empreintes de message sans connaitre les détails de l'implémentation ou les algorithmes qui forment la base de ces concepts. L'indépendance de l'algorithme est réalisée par la définition des types de service cryptographiques « Moteurs » et la définition des classes qui fournissent la fonctionnalité de ces moteurs cryptographiques. Ces classes sont appelées engine classes.

L'indépendance de l'implémentation est réalisée par l'utilisation de l'architecture basée «provider ». Le CSP (Cryptographic Service Provider) se réfère à un package ou un ensemble de packages qui implémentent un ou plusieurs services cryptographiques tels que les algorithmes de signatures numériques et les empreintes de messages. Un programme peut simplement demander un type particulier d'objet (comme un objet de signature) implémentant un service particulier (comme l'algorithme de signature DSA) et obtenir une implémentation d'un des providers installés. Les providers doivent afficher leur mise à jour d'une manière transparente à l'application, par exemple quand des versions plus rapides ou plus sécurisées sont disponibles.

Concepts introduits dans L'API de sécurité

III.5 Les concepts principaux introduits dans l'API de sécurité :

III.5.1 Classes de moteur et les algorithmes

Une classe de moteur (engine class) définit un service cryptographique d'une manière abstraite sans aucune implémentation concrète. Un service cryptographique est généralement associé avec un algorithme particulier qui permet de **[8]**

- fournir des opérations cryptographiques (comme les signatures numériques ou les empreintes de messages).
- générer ou fournir les clés ou les paramètres pour les opérations cryptographiques, ou générer des objets de données (certificats et keystore) qui enveloppent les clés cryptographiques utilisées dans les opérations de cryptographie.

Par exemple Signature et KeyFactory sont deux classes de moteurs. La classe Signature fournit l'accès à la fonctionnalité de l'algorithme de signature numérique. La classe KeyFactory DSA fournit une clé DSA privée ou publique d'un objet de signature DSA, dans un format utilisable respectivement par les méthodes initSign ou initVerify.

JCA/JCE contient les classes du package de sécurité Java liées à la cryptographie, y compris les classes de moteur. Les utilisateurs de l'API utilisent des instances de ces classes de moteur pour réaliser les opérations correspondantes.

L'application interface fournie par la classe de moteur est implémentée en termes de SPI. Pour chaque classe de moteur, il y a une classe SPI abstraite correspondante pour définir les méthodes que le CSP doit implémenter. Pour fournir l'implémentation d'un type particulier de service, pour un algorithme spécifique, un provider doit créer une sous classe de la classe SPI correspondante et fournir l'implémentation de toutes les méthodes de la classe abstraite.

Une instance d'une classe de moteur (l'objet API) encapsule une instance de la classe SPI correspondante (l'objet SPI). L'instance de la classe de moteur (et sa classe SPI correspondante) est crée en appelant la méthode de production getInstance de la classe de moteur. Le nom de chaque classe SPI est le même que celui de sa classe de moteur suivi par Spi. Par exemple, la classe SPI correspondante à la classe de moteur Signature est Signature

Spi. La sous classe SignatureSpi permet de spécifier le type de l'algorithme de signature tel que SHA1withDSA, SHA1with RSA, ou MD5withRSA.

III.5.2 Implémentations et Providers :

Le package du provider JCA par défaut de Java (appelé SUN) inclut les services cryptographiques d'authentification comme les empreintes de message (SHA1) et les signatures numériques (DSA). Le provider Java JCE de Sun (appelé SunJCE) fournit un framework pour le chiffrement, la génération et l'agrément de clé, et les algorithmes MAC-(Message Authentication Code). Le support de chiffrement comprend les chiffrements symétriques et asymétriques, en mode bloc et flux de données (block, and stream cipher) **[8].**

III.6 La classe Provider :

La classe Provider est l'interface des packages des CSPs. Elle a des méthodes pour accéder aux informations de provider (nom, numéro de version et autres informations). Pour fournir des implémentations de services cryptographiques, un groupe de développement écrit le code de l'implémentation et crée une sous classe de la classe Provider. Le constructeur de la sous classe Provider positionne les valeurs des différentes propriétés, l'API de sécurité de Java 2 SDK utilise ces valeurs pour rechercher les services implémentés par ce provider. En d'autres termes, la sous classe spécifie les noms des classes implémentant les services .

Il y a plusieurs types de services qui peuvent être implémentés par les packages du provider. Les implémentations peuvent avoir des caractéristiques différentes. Certaines peuvent être basées programmes, alors que d'autres peuvent être basées matériels. Certaines peuvent être indépendantes de la plateforme, d'autres peuvent être spécifiques à des plateformes. Le code source de certains providers peut être disponible pour révision et évaluation, alors que d'autres providers n'affichent pas leur code source. L'architecture de la cryptographie java laisse aux utilisateurs et aux développeurs de décider de leur besoins.

III.6.1 Comment utiliser les implémentations d'un provider ?

Pour chaque classe de moteur dans l'API, une implémentation particulière est instanciée par l'appel à une méthode de production (factory method) sur la classe de moteur. Une méthode de production est une méthode statique qui retourne une instance d'une classe.

Par exemple, le mécanisme de base pour obtenir un objet Signature est comme suit

```
Signature signature = Signature.getInstance("SHA1withRSA", "BC");
```

La méthode getInstance recherche une implémentation qui satisfait l'algorithme spécifié et les paramètres du provider (son nom, ici BC). Si aucun provider n'est spécifié, getInstance recherche par ordre de préférence, un provider parmi les providers installés, qui a une implémentation de l'algorithme spécifié. Dans la machine virtuelle Java (JVM), les providers sont installés dans un ordre de préférence. Un programme peut avoir la liste de tous les providers installés (en utilisant la méthode getProviders() de la classe Security), et choisir un provider de la liste.

III.6.2 Installation et configuration de providers :

Pour plus d'informations sur l'implémentation d'un provider, voir le guide « How To Implémente a Provider for the Java Cryptographie Architecture » à la page Web:

http://download.oracle.com/javase/6/docs/technotes/guides/security/crypto/HowToImplAPro vider.html

Il ya deux étapes pour installer un provider : installer les classes de package du provider et configurer le provider .

- **Installer les classes du provider :**

Il ya deux façons d'installer les classes du provider :

- Placer le fichier zip ou JAR contenant les classes dans le chemin de la variable d'environnement CLASSPATH.

- Déployer le fichier JAR des classes du provider comme une extension de Java (Package optionnel).

Pour plus d'information sur comment déployer une extension ? voir « How is an extension deployed?» à la page Web :

http://download.oracle.com/javase/6/docs/technotes/guides/extensions/spec.html.

- **Configurer le Provider :**

L'étape suivante est d'ajouter le provider à la liste des providers enregistrés. On doit configurer et enregistrer le provider pour permettre aux clients d'utiliser son service. Il y a deux façons pour enregistrer un provider :

- **Enregistrement statique :** on ajoute le nom du provider à la liste des providers existants dans le fichier java.security du sous répertoire lib/security du JRE. Pour chaque provider disponible il y a une ligne correspondante de la forme :

```
security.provider.n= masterClassName
```

Cette ligne permet de déclarer un provider et spécifier son ordre de préférence n. L'ordre de préférence est celui dans lequel les providers sont recherchés pour les algorithmes demandés (lorsqu'aucun provider n'est spécifié). L'ordre est basé 1 : 1 est le plus préféré, suivi de 2 et ainsi de suite. Master Class Name doit spécifier la classe maitre du provider. Cette classe est toujours une sous classe de la classe Provider. Le constructeur de sous classe positionne les valeurs des propriétés qui sont nécessaires pour l'API de cryptographie Java pour chercher les algorithmes implémentés par le provider.

- **Enregistrement dynamique :** peut être réalisé par l'application client qui appelle des méthodes de la classe Security telle que : Security.addProvider, ou insertProviderAt. Ce type d'enregistrement n'est pas persistant et peut être seulement réalisé par des programmes sûrs et de confiance.

Chaque instance de classe Provider a un nom, un numéro de version, et une chaine décrivant le provider et ses services. La classe Provider a des méthodes pour avoir ces informations, on peut interroger l'instance Provider en appelant les méthodes suivantes : getName(),

getVersion(), et getInfo().

III.7 LE Provider de Bouncy Castle :

III.7.2 Présentation du package crypto de Bouncy Castle (BC) :

Le package crypto de Bouncy Castle est une implémentation d'algorithmes cryptographiques pour la plateforme Java. Les algorithmes du package BC sont conformes au Framework JCE. Lorsque l'accès aux librairies JCE n'est pas possible dans certains cas (comme le cas des applets Java ou les MIDlets Java exécutés sur les petits dispositifs avec ressources limitées), le package BC contient une API légère (lightweight) appropriée a l'environnement mobile comme la plateforme J2ME/MIDP. L'API BC existe aussi en version C# et fournit les même fonctionnalités que celle du Java.

Bouncy Castle est Australien d'origine donc il n'est pas concerné par les restrictions américaines sur l'exportation des logiciels cryptographiques.

III.8 L'environnement de développement intégré NetBeans: NetBeans est aussi une plate-forme générique23 pour le développement d'applications pour stations de travail (bibliothèque Swing (Java)).Elle fournit des ressources pour développer les éléments structurants de ces applications: gestion des menus, des fenêtres configuration, gestion des fichiers, gestion des mises à jour… Des présentations détaillées sont fournies par le centre de documentation de NetBeans .

L'IDE NetBeans comprend toutes les ressources utiles mais il est aussi possible d'installer la plate-forme séparément. Le développement d'applications sur la base de la plate-forme NetBeans consiste en la réalisation de « modules » qui s'insèrent dans la plate-forme et en étendent dynamiquement les fonctions.

Un module est un groupe de classes Java, de portée variée : elle peut consister en une simple classe Java réalisant des fonctions simples (exemple : ajouter une action dans un menu pour éditer le contenu du presse papier) comme elle peut intégrer une application externe complète (exemple : Java profiling suite). Un module peut s'appliquer à l'IDE NetBeans lui-même.

- La réalisation des modules s'appuie sur une API normalisée.
- Un espace de partage entre développeurs est mis en place.

Chapitre IV :

Etude et comparaissent Des algorithmes crypto fournis par le package de Bouncy Castle

Nous avons discuté dans cette partie de l'étude de chaque type de cryptage seul, comme nous l'avons réalisée par rapport à la fin de chaque type d'accès. S'il vous plaît objectif de cette étude en général , Lorsque nous avons divisé le travail en cinq fenêtres comme suit: Fenêtre principale comme un point d'entrée pour le programme et une fenêtre secondaire à choisir un travail de chiffrement à faire en termes émergent de chaque bouton un type spécifique de type réfléchie de cryptage.

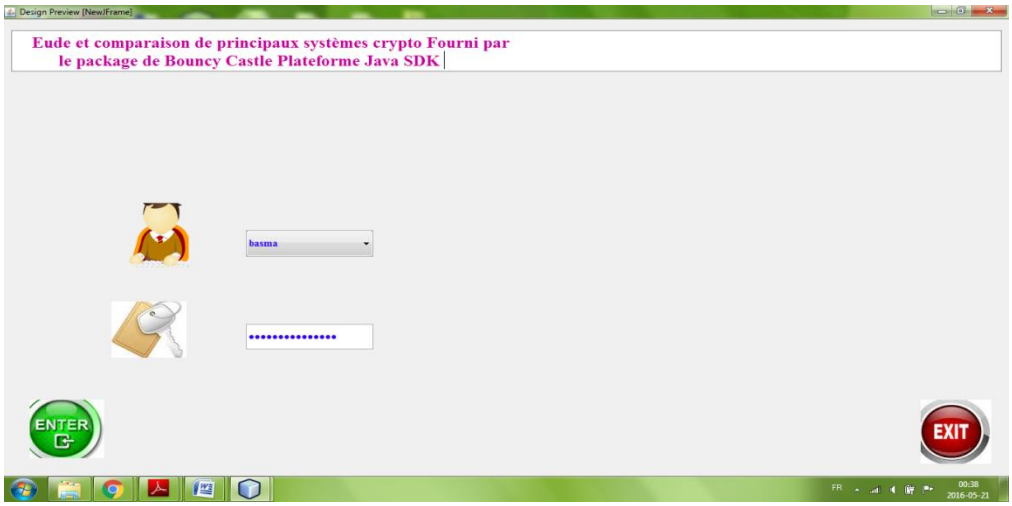

Figure IV.1: Interface principale

Figure IV.2: Interface secondaire

IV .1 Le Test de Bouncy Castle :

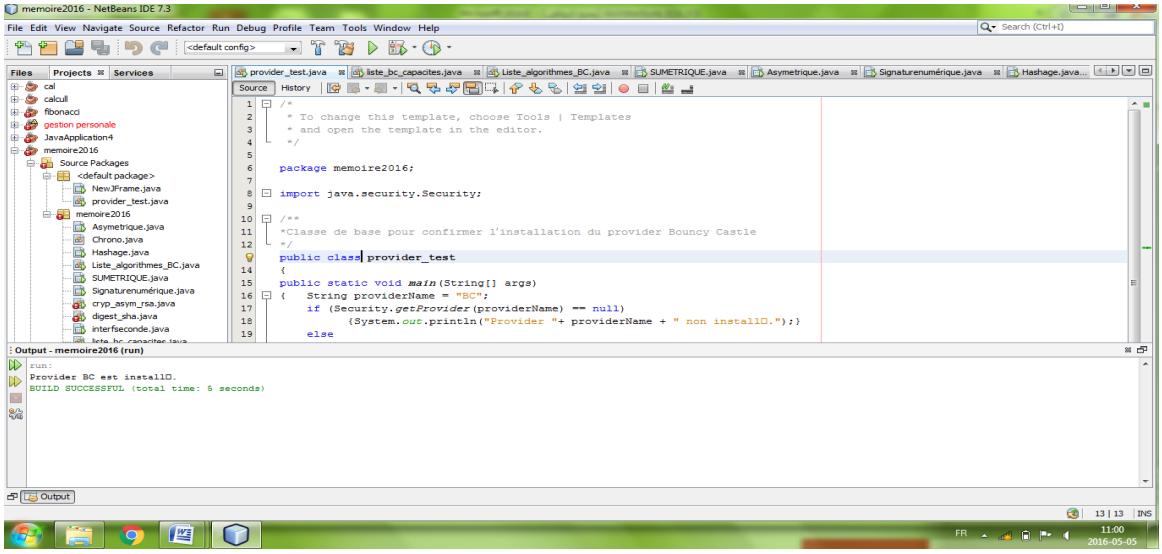

Figure IV.3 Test de Bouncy Castle

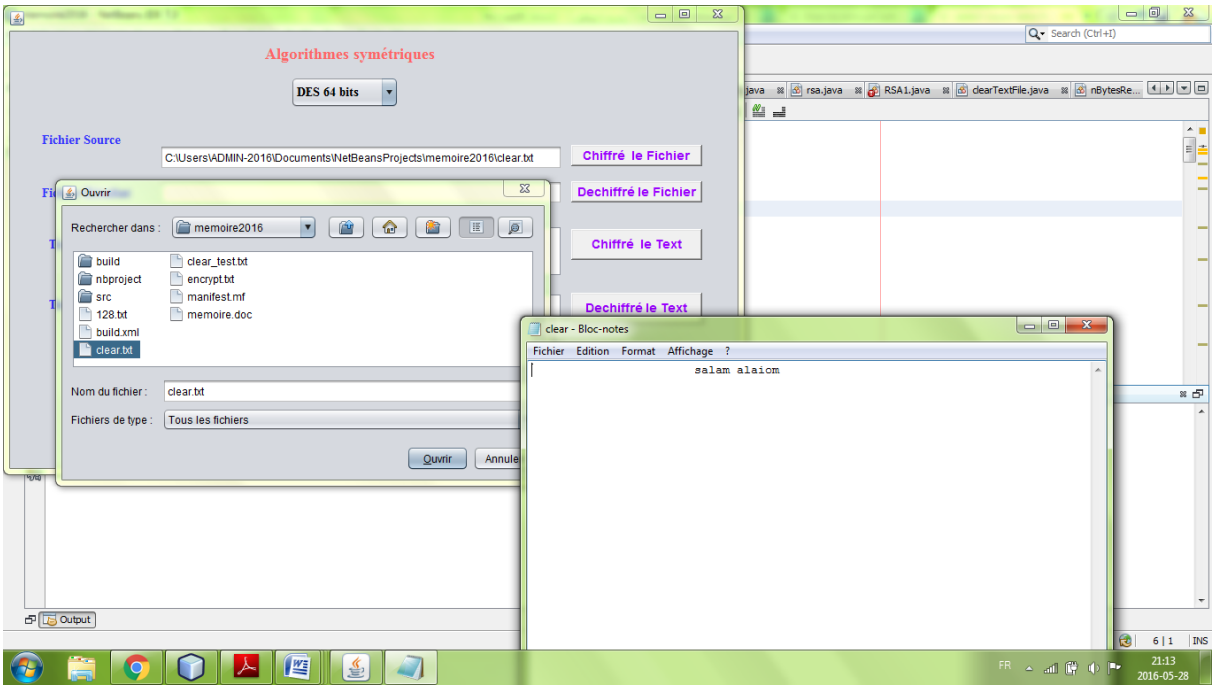

Figure IV.4: Algorithmes symétriques

IV .2 Algorithmes de chiffrement symétriques

Dans ce type de cryptage que nous avons choisi les deux: dans Block cipher : DES (64 bits) 3DES (128,256 bits) , AES (128.,192,256 bits) et dans Steam cipher : RC2 , RC4

IV .2 .1 Comparé le tempe d'exécution de Chiffré et Déchiffré les Fichiers différent tailles en utilisant les algorithmes de Symétrique dans la Cryptage moderne:

Dans ce tableau, nous avons calculé la moyen de tempe (ms) de récurrence de chiffrement ou de déchiffrement 10 fois .

La taille	L'opération	DES	3DES		AES		
		64 bits	128 bits	256 bits	128 bits	192 bits	256 bits
1 MO	Chiffrement	2545.9ms	1312 ms	1311.9 ms	1295 ms	1313.5 ms	1321.3 ms
	Déchiffrement	535.1 ms	1330.7 ms	1352.5 ms	1280 ms	1349.4 ms	1325.9 ms
2 MO	Chiffrement	5090.2 ms	2563.1 ms	2636.4 ms	2543 ms	2687.8 ms	2687.9 ms
	Déchiffrement	5049.7 ms	2218.3 ms	2260.5 ms	2621 ms	535.1 ms	1113.9 ms
3 MO	Chiffrement	7726.7 ms	3310.3 ms	3354 ms	3167 ms	3213.6 ms	3261.9 ms
	Déchiffrement	1566.3 ms	1984.3 ms	1260.6 ms	3962 ms	1577.1 ms	1600.5 ms

Table IV.1: Comparaissent les Fichiers

IV .2 .2 Comparé le tempe d'exécution de Chiffré et Déchiffré le texte en utilisant les algorithmes de Symétrique dans la Cryptage moderne:

Dans ce tableau nous avons calculé la moyen de tempe (ms) de récurrence de chiffrement ou de déchiffrement 10 fois .

L'opération	DES	3DES		AES		
	64 bits	128 bits	256 bits	128 bits	192 bits	256 bits
Chiffrement	531.9 ms	431.9 ms	438.4 ms	441.7 ms	453.9 ms	435.2 ms
Déchiffrement	460.2 ms	447 ms	437.2 ms	444.8 ms	438.4 ms	436.8 ms

Table IV.2: Comparaissent de les textes

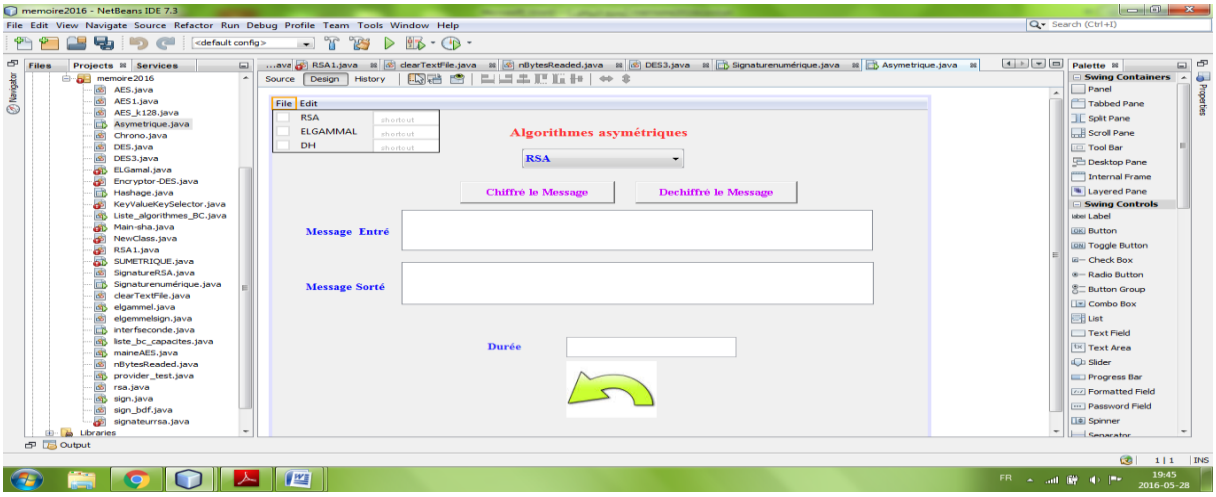

Figure IV.5: Algorithmes Asymétriques

IV .3 Algorithmes de chiffrement Asymétriques :

Dans ce type de cryptage que nous avons choisi les deux: RSA (128,192 bits) , DH ELGAMMAL

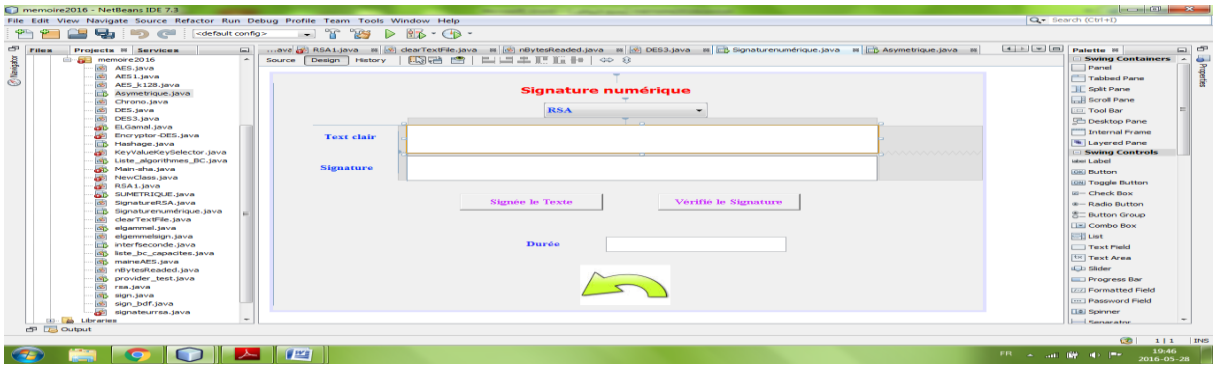

Figure IV.6: Signature numérique

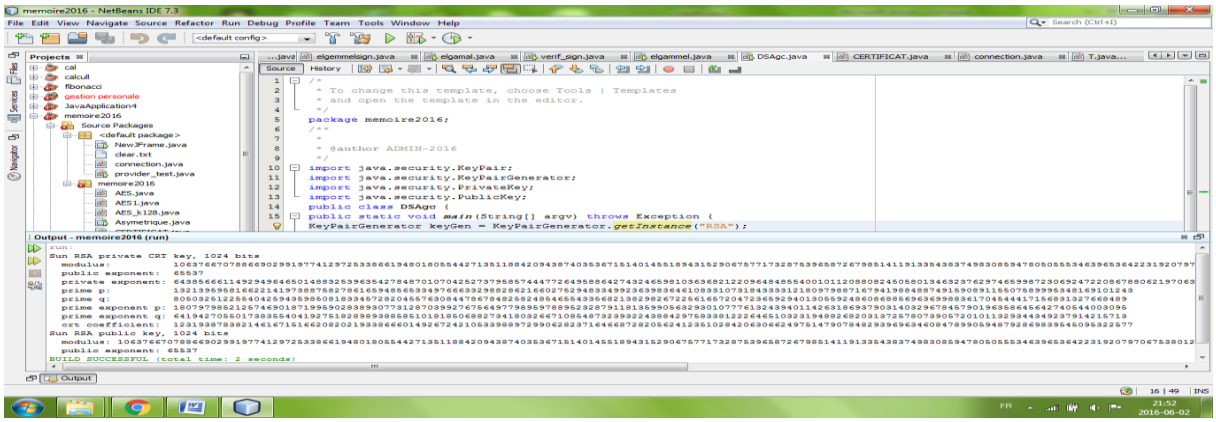

Figure IV.7: Génération de Pair de clé : algorithmes RSA

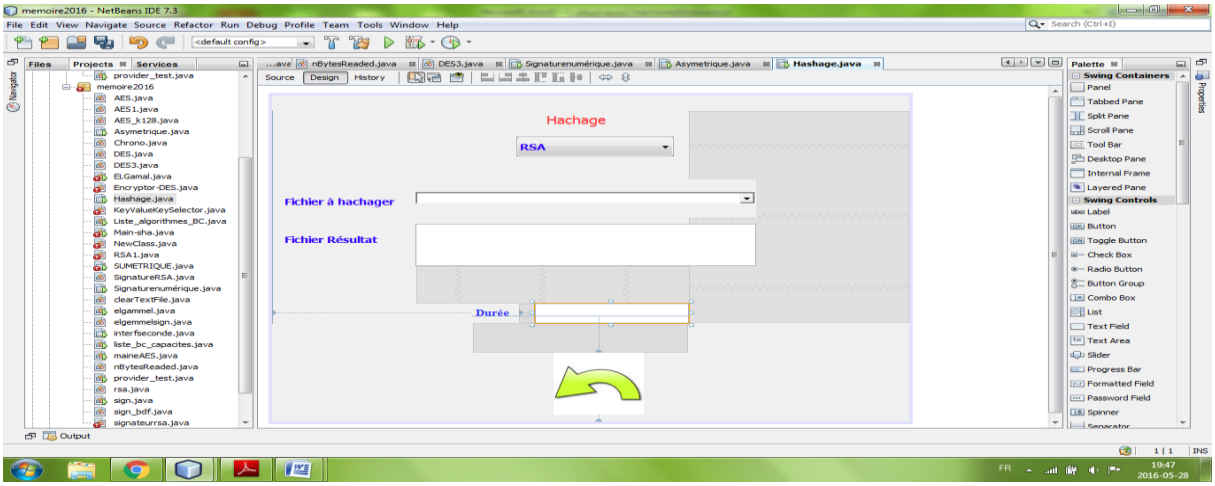

Figure IV.8: Fonction de Hachage

IV.4 Fonctions de hachage et Macs :

Dans ce type de cryptage que nous avons choisi les deux: RSA , DSA , ECDSA dans Signature numérique et SHA1, SHA (256 , 512 bits) , MD5, RIPEMD (256,512 bits) dans fonction de hachage

Conclusion générale

Les cryptographes n'ont cessé de redoubler d'ingéniosité, faisant se succéder des dizaines de systèmes de chiffrement plus recherchés les uns que les autres. Se livrant bataille pour la gloire ou l'argent, ils n'ont cessé de faire évoluer cette science qu'est la cryptographie. Avec d'abord une mécanisation (notamment la machine Enigma), puis grâce à l'avènement des ordinateurs, et avec eux une puissance de calcul surpassant de loin le niveau humain, la cryptographie a su trouver son chemin dans les dédales du progrès. Progrès qui ne s'arrête jamais ! Déjà maintenant, d'autres voies, encore obscures, se dessinent

Les ordinateurs quantiques : ils procureraient une puissance de calcul colossale, et une sécurité infaillible ! Cependant, la théorie est là, mais la mise en œuvre semble très difficile... pour l'instant :) la cryptographie multi variable quadratique : des multiples équations faisant intervenir jusqu'à 120 variables.

Le chaos chiffrant : "noyer" le message dans un signal chaotique et, connaissant les caractéristiques du signal, le retrouver. Des méthodes fondées sur les courbes elliptiques .

Reference :

[1] Omar Cheikhrouhou, Les Algorithmes Cryptographiques Symétriques

[2] Nitaj, Abderrahmane. CRYPTANALYSE DE RSA. Laboratoire de Mathématiques

Nicolas Oresme, Université de Caen, France.

[3] Niels Ferguson, Bruce Schneier. Practical Cryptography. Indiana: Wiley Publishing,.

[4] Lo Johnny Li-Chang. A framework for cryptography algorithms on mobile devices Faculty of Engineering, Build and Information Technology Department of Computer Science UNIVERSITY OF PRETORIA

[5] Stephen A. Thomas, SSL & TLS Essentials Securing the web, Appendix A: X.509 Certificates, Wiley Computer Publishing, John Wiley & Sons, Inc

[6] RSA Laboratories, PKCS#1 v2.1: RSA Cryptography Standard, PKCS Standards.

[7] On the Web sun.com, White Paper Java Security Overview, Sun Microsystems.

[8] Oracle Java SE Documentation, Java ™ Cryptography Architecture (JCA) Référence Guide for Java TM Platform Standard Edition 6 .

[9] Hook, David. Beginning Cryptography with Java (Chapter 1 :The JCA and the JCE) .

[10] http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf

[11]http://www.apprendre-en-ligne.net/crypto/histoire

[12] http://www.primenumbers.net/Renaud/fr/crypto/PrincipesCrypto.htm

[13] www.clusir-rha.rf/sites/de faut /files/upload/Lyon/SSi/ClusirCryptographie.pdf

[14]https://www.google.dz/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q La+cryptographie+classique

[15]http://www-igm.univ-mlv.fr/~dr/XPOSE2001/amos/cryptographie.asp?http://www-igm.univ-mlv.fr/~dr/XPOSE2001/amos/cryptages.htm

[16]http://ser-info nantes.fr/users/info3/weblog/c6a6e/Principes_de_la_cryptographie_moderne.html

[17] https://www.securiteinfo.com/cryptographie/aes.shtml

[18] https://fr.wikipedia.org/wiki/Cryptographie_hybride

[19] http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

ملخص :

هذا العمل هو مقارنة خوارزميات التشفير الرئيسية التي يوفرها **Bouncy Castle** لتمكين مطوري التطبيقات لاستخدام خوارزميات التشفير الأكثر فعالية في إدارة أمن تكنولوجيا المعلومات في طلباتهم لتحقيقه .

Le Bouncy Castle : هو مجموعة من واجهات برمجة التطبيقات(**API**) المستخدمة في الترميز، ويشمل واجهات برمجة التطبيقات لكل من **Java JCE / JCA** ، **C #**،كما يتناسب مع البيئات المتنقلة وتطبيقات **Java** .

لقد اخترنا **Bouncy Castle** لأنه مفتوح المصدر، وغني بخوارزميات التشفير و هو في تطوير مستمر منذ ظهوره على شبكة الإنترنت.

نحن نستخدم معايير لقياس خوارزميات التشفير هذه والمعايير الرئيسية هي سلامة وسرعة الحساب بحيث يتم قياس الأمن في عدد من البتات التي تشكل المفتاح ، يتم قياس سرعة الحساب في وقت التنفيذ في ميلي ثانية أو بايت في الثانية

الكلمات الرئيسية: علم التشفير ، حزمة التشفير, **Le Bouncy Castle** ، معيار التشفير ، الأمن، **Java JCA/JCE**، تشفير ، التوقيع الرقمي .

Résumé :

Ce travail consiste à comparer les principaux algorithmes crypto fournis par le package crypto open source de Bouncy Castle pour permettre aux développeurs d'applications d'utiliser les algorithmes crypto les plus efficaces pour la prise en charge de la sécurité informatique dans leurs applications à réaliser.

Bouncy Castle est une collection d'API utilisées en cryptographie. Elle comprend les API tant pour le Java JCA/JCE, C#, et une API légère adaptée aux environnements mobiles et les applets Java. Nous avons choisi ce package parce qu'il est open source, riche en algorithmes crypto et connait un développement continu depuis son apparition sur Internet . Ce package ressemble beaucoup à la librairie C openssl qui est conforme aux différents standards en vigueur.

Nous utilisons des Benchmarks crypto pour mesurer ces algorithmes, Les principaux benchmarks sont la sécurité et la rapidité de calcul. La sécurité se mesure en nombre de bits formant la clé, la rapidité de calcul se mesure en temps d'exécution en millisecondes ou en nombre d'octets par seconde.

Les mots clé : Cryptographie, package crypto, Bouncy Castle, Benchmark Crypto, sécurité Java JCA/JCE, chiffrement, signature numérique.

Summary :

This work is to compare crypto algorithms provided by the Bouncy Castle crypto open source package to enable application developers to use crypto algorithms most effective in the management of IT security in their applications to achieve.

Bouncy Castle: is a collection of APIs used in cryptography. He include APIs for both Java JCA / JCE C #, and a slight API suited to mobile environments and Java applets. We chose this package because it is open source, rich in crypto algorithms and suffers a continuous development since its appearance on the Internet. This package is much like the C library openssl which complies with different standards.

We use Benchmarks to measure these crypto algorithms, Key benchmarks are safety and speed of calculation. Security is measured in number of bits forming the key, the calculation speed is measured in execution time in milliseconds or number of bytes per second.

Key words: Cryptography, crypto package, Bouncy Castle Crypto Benchmark, security Java JCA / JCE, encryptions, digital signature.