**République Algérienne Démocratique et Populaire**
**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**

**Université A/Mira de Béjaïa**
**Faculté des Sciences Exactes**
**Département d'Informatique**

# *Mémoire de fin de cycle*

**En vue de l'obtention d'un Master en Réseaux et Systèmes Distribués**

**Option**

***Recherche***

## Thème

# Algorithmes de chiffrement asymétrique à base de factorisation : mise en œuvre de RSA, Rabin et Paillier.

**Pésenté par :**

**M^me^ IDIRI Fahima.**

Devant le jury composé de :

| | | | |
|---|---|---|---|
| **Président** | **M^r^ D.TOUAZI** | **M.A.A** | **Université de Béjaïa.** |
| **Examinatrice** | **M^me^ R. BELALETA** | **M.A.B** | **Université de Béjaïa.** |
| **Examinatrice** | **M^me^ A. HOUHA** | **M.A.B** | **Université de Béjaïa.** |
| **Promoteur** | **M^r^ M. OMAR** | **M.C.A** | **Université de Béjaïa.** |
| **Co-promoteur** | **M^r^ K. AMROUN** | **M.C.A** | **Université de Béjaïa.** |

Promotion 2015-2016.

# ⁕ *Remerciements* ⁕

Je tien à remercier mon tuteur M$^{r}$ M. OMAR, Maitre de Conférences à l'université de Béjaïa, de m'avoir proposer le thème de ce mémoire, pour l'aide et le temps qu'il m'a consacré.

Mes remerciements s'adressent également à M$^{r}$ D. TOUAZI, Maître assistant à l'université de Béjaïa d'avoir accepté de présider le jury et à M$^{me}$ A. HOUHA, Maître assistante à l'université de Béjaïa, R. BELALTA, Maître assistante à l'université de Béjaïa d'avoir accepté d'examiner ce travail.

J'exprime ma gratitude à M$^{r}$ R. ouzeggane, Maître assistant à l'université de Béjaïa d'avoir accepté de répondre à mes questions avec une grande compréhension et générosité et pour tous ses conseils thecniques.

Je n'oublie pas mes parents et mon mari pour leur contribution, leur soutien et leur patience.

Enfin, j'adresse mes plus sincères remerciements à tous mes proches et amis, qui m'ont toujours soutenue et encouragée au cours de la réalisation de ce mémoire.

Merci à toutes et à tous.

# ⁕ *Dédicaces* ⁕

***A ma très chère mère,***

Douce, affable, honorable, aimable : tu représente pour moi le symbole de la bonté par exellence, la source de tendresse et l'exemple de dévouement qui n'a pas césser de m'encourager et de prier pour moi.

Tes invocations et bénidiction m'ont étées d'un grand secours pour mener à bien mes études.

aucune dédicaces ne saurai être assez éloquente pour exprimer ce que tu mérites pour tous les sacrifices que tu n'a césser de me donner depuis ma naissance, durant mon enfance et même à l'âge adulte.

Tu a fais plus qu'une mère puisse faire pour que ses enfants suivent le bon chemein dans leur vie et leurs études.

Je te dédie ce travail en témoignage de mon profond amour. Puisse Dieu, le tout puissant, te pésérver, t'accorder santé, longue vie et boheur.

***A mon très cher papa,***

Aucune dédicace ne saurai exprimer l'amour, l'estime, le dévouement et le respet que j'ai toujours eu à votre égard.

Rien au monde ne vaut les efforts fournis jours et nuits pour mon éducation et mon bien être.

Ce travail est le fruit de tes sacrifices que tu a consentis pour mon éducation et ma formation.

***A mon très cher mari Amine,***

Mon âme soeur et la lumière de mon chemin.

A tes cotés, ma vie est remplie de belles surprises.

Tes sacrifices, ton soutien moral et matériel, ta douceur, ton amour et gentiellesse sans égal, m'ont donnés force pour aller droit devant et réussire mes études.

Que dieu réunisse nos chemins pour un long commun serein et que ce travail soit témoignage de ma reconnaissance et de mon amour sincère.

***A mes très chères soeurs et très cher frères,***

***A ma très chère belle famille,***

***A mon bou de chou Youssouf.***

***Fahima***

# TABLE DES MATIÈRES

# TABLE DES FIGURES

# LISTES DES ABRÉVIATIONS

- **DES** : Data Encryption Standard.
- **NSA** : National Security Agency.
- **IBM** : International Business Machines.
- **IDE** : Integrated Development Environment.

# INTRODUCTION GÉNÉRALE

A l'heure actuelle, les besoins en matière de sécurité sont grandissants, et la tendance n'est certainement pas à la baisse. Mais pourquoi ? Tout d'abord parce que le matériel informatique est omniprésent : les sociétés sont devenus dépendantes de leur outil informatique.

En effet, d'une part le matériel est accessible à un prix très abordable, et d'autre part, les logiciels tendent à se simplifier (au niveau de l'utilisation !) et permettent une prise en main rapide.

D'un autre coté, les entreprises, elles aussi informatisées, nécessitent un réseau sécurisé pour le transfert des données, que ce soit entre les machines de cette entreprise, ou avec des machines externes, distantes de plusieurs milliers de kilomètres.

La sécurité recouvre l'ensemble de techniques informatiques permettant de réduire au maximum les chances de fuites d'information, de modification de données ou de détérioration des services. Elle consiste à un très grand nombre de méthodes, de technologies et d'architectures permettant d'atteindre un certain niveau de protection.
Si on l'observe d'une manière plus générale, elle est d'ailleurs présente à plusieurs niveaux des différentes portées de l'information (Données, application, machine, réseaux). Du coup les menaces pour les libertés professionnelles et individuelles sont réelles, par conséquent elle devient primordiale et un des éléments fondamentaux à connaitre pour préserver et l'outil de travail que constituent nos bases de données et la confidentialité à laquelle on a droit.

Pour toutes ces raisons, le recours à la cryptographie, une technologie essentielle dans la sécurité, s'impose dès qu'une opération symbolique peut être détournée de son objet.
La cryptographie se scinde en deux parties nettement différenciées :

- d'une part la cryptographie à clef secrète, encore appelée symétrique ou bien classique ;
- d'autre part la cryptographie à clef publique, dite également asymétrique ou moderne.

Nous allons nous intéresser à la cryptographie asymétrique, nous allons étudier des algorithmes de chiffrements asymétrique qui tirent leur sécurité du problème de factorisation. Ce rapport est structurer en quatre chapitres : le premier chapitre donne un aperçu sur la cryptographie, le deuxième aborde le chiffrement à clé publique et quelques algorithmes de chiffrement basé sur le problème de factorisation.
le troisième et le quatrième chapitres, quant à eux sont consacrés à l'implémentation de ces algorithmes. Enfin nous allons conclure par quelques perspectives concernant l'implémentation des algorithmes de chiffrement.

# CHAPITRE 1

# CRYPTOGRAPHIE

## 1.1 Introduction

Il est difficile de réduire tout un large secteur d'activité scientifique à une formule, et de ce fait une bonne définition de la cryptographie ne pourra s'acquérir qu'au fur et à mesure de son étude et de sa pratique. En effet, ce chapitre abordera certaines notions principales de la cryptographie à savoir les méthodes de chiffrement. Nous nous tournerons dans un premier temps vers l'histoire de la cryptographie, ensuite on verra la cryptographie symétrique.

## 1.2 Un tour d'horizon

Dès que les hommes apprirent à communiquer, ils durent trouver des moyens d'assurer la confidentialité d'une partie de leurs communications : l'origine de la cryptographie remonte sans doute aux origines de l'homme.
En effet, le mot cryptographie est un terme générique désignant l'ensemble des techniques permettant de chiffrer des messages, c'est-à-dire de les rendre inintelligibles sans une action spécifique.
Du bâton nommé "Scytale" au VIIe siècle avant JC, en passant par le carré de "Polybe" ou encore "le code de César", on assista au développement plus ou moins ingénieux de techniques de chiffrement expérimentales dont la sécurité reposait essentiellement dans la confiance que leur accordaient leurs utilisateurs. Après la première guerre mondiale a lieu une première révolution technologique.
Mais ce n'est qu'à l'avènement de l'informatique et d'Internet que la cryptographie prend tout son sens. Les efforts conjoints d'IBM et de la NSA conduisent à l'élaboration du

DES, l'algorithme de chiffrement le plus utilisé au monde durant le dernier quart du XXème siècle.
A l'ère d'Internet, le nombre d'applications civiles de chiffrement(banques, télécommunications, cartes bleues, etc.) explose. Le besoin d'apporter une sécurité accrue dans les transactions électroniques font naître les notions de signature et authentification électronique. La première technique de chiffrement à clef publique sûre (intimement liée à ces notions) apparaît : le RSA [1, 2, 3].

## 1.3 Chiffrement à clé secrète

Dans un système de chiffrement à clé secrète ou symétrique ou encore conventionnelle un expéditeur et un destinataire partage une même clé secrète, cette clé est utilisée à la fois pour le chiffrement et pour le déchiffrement et doit rester secrète de tout observateur ennemi. A chaque clé K sont associées une fonction de chiffrement $C_K$ et une fonction de déchiffrement $D_K$. L'expéditeur chiffre le texte clair m pour obtenir le texte chiffré $c = C_K(m)$ et envoie c au destinataire. Le destinataire rétablit le texte en clair m en calculant $m = D_K(c)$ (voir la figure 1.1) [1, 2, 3].

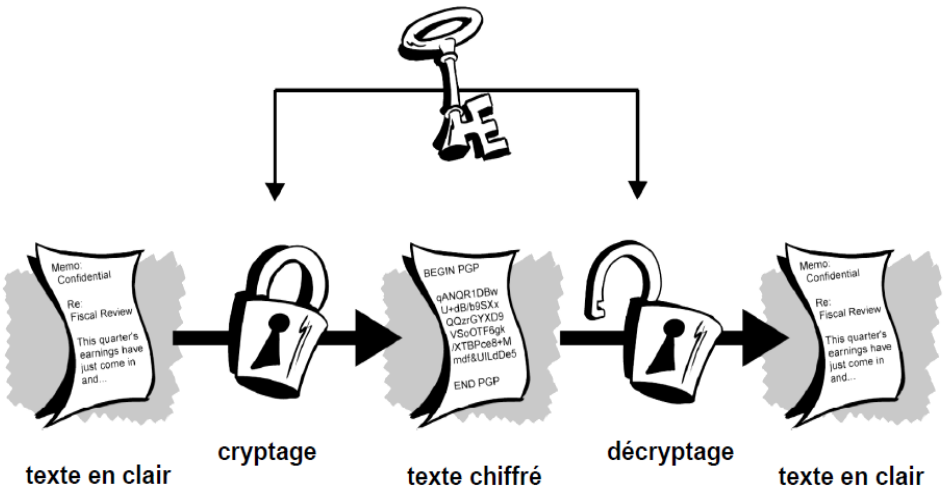



FIGURE 1.1 – Chiffrement symétrique [3].

Les cryptosystèmes symétriques se répartissent en deux familles : le chiffrement à flot et le chiffrement par bloc.

### 1.3.1 Chiffrement à flot

Une méthode de chiffrement à flot opère individuellement sur chaque bit de texte clair en utilisant une transformation qui varie en fonction de la place du bit en entré.
Le cryptosystème de Vermam appelé aussi one-time-pad ou encore masque jetable est le prototype de ces systèmes. il utilise une clé secrète très longue qui devrait de manière idéale représenter une suite aléatoire de bits. Si on a un message m de n bits à chiffrer, on considère les n premiers bits de la clé qui constituent un mot K et on calcul le "ou exclusif bit à bit" entre le message et cette partie de la clé. Ainsi la partie K de la clé sert de masque. Le destinataire qui partage la même clé extrait de la même façon la partie K et récupère alors le texte clair m en calculant $m = c \oplus K$. Les deux interlocuteurs jettent la partie K utilisée et peuvent effectuer une nouvelle transaction en procédant de même avec le reste de la clé [3, 1].

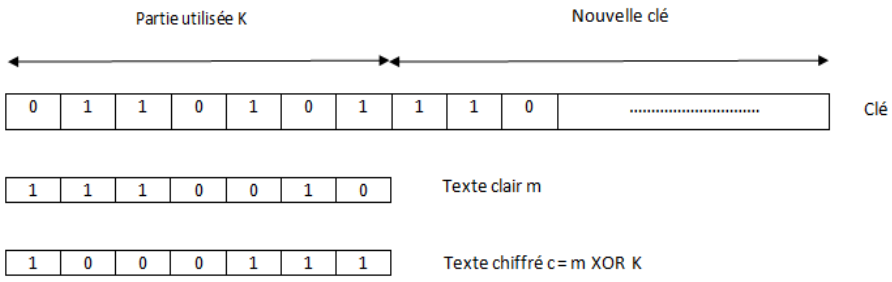



FIGURE 1.2 – Le masque jetable [3].

### 1.3.2 Chiffrement par bloc

Un système de chiffrement par bloc opère avec une transformation fixe qui s'applique sur des blocs de texte clair, de taille fixe (la taille de bloc est comprise entre 32 et 512 bits) [3, 1].

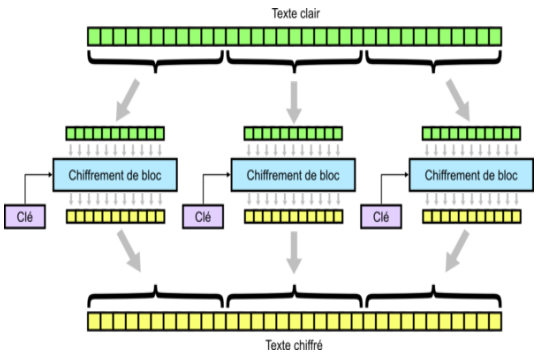


FIGURE 1.3 – Chiffrement par bloc [3].

Il existe de nombreux systèmes de chiffrement par bloc, citons quelques systèmes [1] :
- DES (Data Encryption Standard).
- 3-DES (ou triple DES).
- MISTY1.
- IDEA (International Data Encryption Algorithm).
- AES (Advanced Encryption Standard).
- Camellia.
- SHACAL-2.

## 1.4 Conclusion

Ce chapitre a pour objectif d'introduire la notion de cryptographie et une de ses techniques, qu'est le chiffrement symétrique. Dans le prochain chapitre on va établire un état de l'art sur les techniques de chiffrement asymétrique à base de factorisation.

# CHAPITRE 2

# CHIFFREMENT ASYMÉTRIQUE

## 2.1 Introduction

Le sujet traditionnel de la cryptographie est le chiffrement. Chiffrer sert à garder secrets des messages ou des données enregistrées. Dans ce chapitre, nous introduisons le chiffrement à clé publique, ensuite on vas étudier les principaux algorithmes de chiffrement asymétriques qui tirent leur sécurité du problème de factorisation, à savoir RSA, Rabin Micheal, Paillier, Guillou Quisquater, Goldwasser Micali.

## 2.2 Chiffrement à clé publique

le principe du chiffrement à clé publique (appelé aussi chiffrement asymétrique) est apparu en 1976. c'est une méthode de chiffrement qui s'oppose au chiffrement symétrique. Elle repose sur l'utilisation d'une bi-clé, constitué d'une clé publique(qui est diffusée) et d'une clé privée (gardée secrète), l'une permettant de coder le message et l'autre de le décoder (voir la figure 2.1) [1, 2, 3].

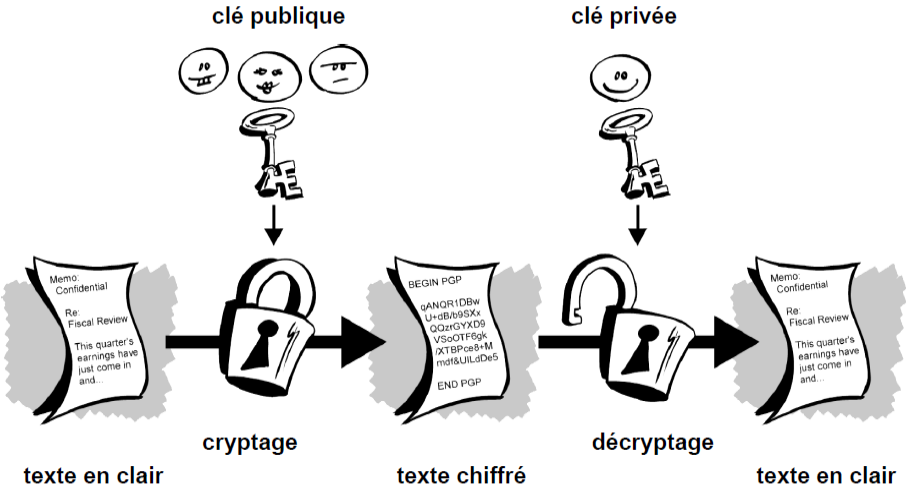



FIGURE 2.1 – Chiffrement asymétrique [1].

Voici quelques algorithmes de chiffrement asymétrique basés sur le problème de factorisation :

### 2.2.1 RSA

Le fonctionnement du cryptosystème RSA est basé sur la difficulté de factoriser de grands entiers. Les étapes de cet algorithme sont [1, 2, 3, 8, 9] :

- Génération de clés :

· Choisir p et q, deux nombres premiers distincts, $p \neq q$.
· calculer leur produit $n = p \cdot q$, appelé module de chiffrement.
· calculer $\varphi(n) = (p-1) \cdot (q-1)$(c'est la valeur de l'indicatrice d'Euler en n).
· choisir un entier naturel e premier avec $\varphi(n)$ tel que : $1 < e < \varphi(n)$ et $pgcd(e, \varphi(n)) = 1$, appelé exposant de chiffrement.
· calculer l'entier naturel d, inverse de e modulo $\varphi(n)$ et strictement inférieur à $\varphi(n)$, appelé exposant de déchiffrement ; d peut se calculer efficacement par l'algorithme d'Euclide étendu.
· (e,n) est la clé publique et (d,n) est la clé privée.

- Chiffrement :
  Si M est un entier naturel strictement inférieur à n représentant un message, alors le message chiffré sera représenté par :
  $C = M^e \bmod n$

- Déchiffrement :
  Pour déchiffrer C, on utilise d, l'inverse de $(e \bmod (p-1)\cdot(q-1))$, et on retrouve le message clair M par $M = C^d \bmod n$.

- Complexité :

Le chiffrement et le déchiffrement de RSA nécessitent une exponentiation modulo n. La taille de (d,e) est celle de $\varphi(n)$, soit celle de n puisque $\varphi(n) = (p-1)\cdot(q-1)$. Cela coûtent $O(Log(n))^3$ multiplications modulo n.

### 2.2.2 Rabin Micheal

Le cryptosystème de Rabin est un cryptosystème asymétrique basé sur la difficulté du problème de factorisation (comme RSA). Il a été inventé en 1979 par Michael Rabin : c'est le premier cryptosystème asymétrique dont la sécurité se réduit à l'intractabilité de la factorisation d'un nombre entier.
Le cryptosystème de Rabin a l'avantage de disposer d'une preuve de difficulté aussi grande que la factorisation d'entiers, preuve qui n'existe pas encore pour RSA. Il a par contre un désavantage dû à un non-déterminisme : une sortie produite par la fonction présente dans le cryptosystème peut être le résultat de quatre entrées distinctes. Il faut donc déterminer quelle entrée est la bonne par un mécanisme annexe [1, 2, 3, 8, 9]. Les étapes de cet algorithmes sont :

- Génération de clés :
  · Choisir deux grands nombres premiers, p et q, au hasard, tels que $p \bmod 4 = 3$ et $q \bmod 4 = 3$.
  · Posons $n = p \cdot q$, ce qui fait de n la clé publique. Les nombres premiers p et q constituent la clé privée.

Pour chiffrer, on n'a besoin que de la clé publique, n. Pour déchiffrer, les facteurs de n, p et q, sont nécessaires.

- Chiffrement :

Pour le chiffrement, seule la clé publique, n, est utilisée. On produit le texte chiffré à partir du texte en clair m comme suit : soit $P = (0, ..., n-1)$ l'espace des textes en clair possibles (tous des nombres) et posons $m \in P$ comme étant le texte en clair. Le texte chiffré c se détermine comme suit :

$$C = m^2 \bmod n.$$

Autrement dit, c est le résidu quadratique du carré du texte en clair, pris modulo n. En pratique, le chiffrement par bloc est généralement utilisé.

- Déchiffrement :

Pour déchiffrer, la clé privée est nécessaire. Le processus est comme suit :

· On calcule les racines carrées $m_p = C^{(p+1)/4} \bmod p$ et $m_q = C^{(q+1)/4} \bmod q$
· On calcule les quatre racines carrées $+r, -r, +s$ et $-s$ de c :

$r = (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \bmod n.$
$-r = n - r.$
$s = (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \bmod n.$
$-s = n - s.$

- Complexité :

La complexité d'algorithme de Rabin Michael est $O(log(n))^2$. Le seul calcul coûteux de cet algorithme est le calcul de multiplications modulo n et de la puissance $m^2 \bmod n$ (à effectuer au plus k fois).

### 2.2.3 Goldwasser Micali

le cryptosystème de Goldwasser-Micali (GM) est un algorithme de chiffrement asymétrique, développé par Shafi Goldwasser et Silvio Micali en 1982. GM est le premier cryptosystème à chiffrement probabiliste qui est prouvablement sûr, avec des hypothèses cryptographiques standards. il n'est pas efficace vue que les textes chiffrés peuvent être des centaines de fois plus longs que les textes d'origine :il produit une valeur de grandeur approximative de $|N|$ pour chiffer un seul bit de message. Afin de prouver la sécurité de ce cryptosystème, ses auteurs ont proposé la définition de sécurité sémantique qui est, de nos jours, largement utilisée.
La preuve que le cryptosystème GM est sémantiquement sûr est basée sur l'hypothèse d'intractabilité du problème de la résiduosité quadratique modulo N où $n = p \cdot q$, avec p,q de grands nombres premiers. En bref, ce problème se résout facilement lorsque la factorisation de N est connue [1, 2, 3, 8, 9].

- Génération de clés :

· Sélectionner deux grands nombres premiers p et q aléatoirement tel que $p \neq q$.
· Calculer $n = p \cdot q$.
· Sélectionner $z \in Z_n$ tel que z est un résidu quadratique non modulo n et le symbole de Jacobi : $(z/n = 1)$.
· La clé publique est (n,z), et la clé privée est la paire (p,q).

- Chiffrement :

Pour que B peut chiffrer un message m pour A, il doit faire ce qui suit :

· obtenir une clé publique (n,z).

· Représenter le message m comme une chaîne binaire $m = m_1 m_2 ... m_t$ de longueur t.

· Pour i allant de 1 à t faire :

1. Choisir un $r_i \in Z_n$ au hazard.
2. Si $m_{i=1}$ alors $c_i \longrightarrow z \cdot r_i^2 \bmod n$ ; sinon $c_i \longrightarrow r_i^2 \bmod n$.

· Envoyer le t-uplet $c = c_1, c_2, ..., c_t$ à A.

- Déchiffrement :

Pour récupérer le clair m de c, A doit faire ce qui suit :

· Pour i allant de 1 à t faire :

1. Calculer le symbole de Legendre $e_i = (c_i/p)$.
2. Si $e_{i=1}$ alors $m_i \longrightarrow 0$ ; sinon $m_i \longrightarrow 1$.

· Le message déchiffré est $m = m_1 m_2 ... m_t$.

- Complexité :

Pour calculer la complexité de cet algorithme, on calcule la complexité de chiffrement et de déchiffrement, tel que :

∗ La complexité de chiffrement est celle des opérations de la boucle de l'étape (b) de l'algorithme de chiffrement qui est égale à $O(log(n))^3$.

∗ De même pour le déchiffrement, tel que sa complexité dépend des opérations de la boucle (a) de l'algorithme de déchiffrement et qui est égale à la complexité de l'opération $e_i = C_i/p$. On constate alors qu'elle est égale à $O(log(n))^2$.

Nous concluons que le chiffrement et le déchiffrement de Goldwasser-Micali coutent $O(log(n))^2$ multiplications modulo n.

### 2.2.4 Guillou Quisquater

Guillou-Quisquater ou GQ est un protocole d'authentification numérique inventé par Louis Guillou et Jean-Jacques Quisquater. Il est basé sur le problème de factorisation. Il utilise une exponentiation de RSA, et est utilisé dans nos jours dans les cartes à puces.
L'algorithme fonctionne de la façon suivante : soit une personne A qui veut s'authentifier auprès d'une autre personne B, A possède un certificat public $J_a$ ainsi qu'un certificat privé $S_a = J_a^{-s} mod(n)$. Le but est de prouver la possession du certificat privé [1, 2, 3, 8, 9].

- Génération de clés :
· $n = p \cdot q$, de la même manière que dans RSA.
· un nombre $v$ qui sert de clé publique tel que $pgcd(v, \varphi(n)) = 1$.
· un nombre $s$ qui sert de clé privée tel que $s \cdot v = 1 \ mod \ \varphi(n)$.

- Chiffrement :
· A choisit un nombre aléatoire $r$.
· A calcule $x = r^v \bmod n$.
· A envoie $x$ et $J_a$ à B.
· B choisit un nombre aléatoire e tel que $1 \leq e \leq v$.
· B envoie e à A.
· A calcule $y = r \cdot S_a^e \bmod n$ et l'envoie à B.

- Déchiffrement :
· B calcul $J_a^e \cdot y^v$ et vérifie que le résultat est égal à $x$ et différent de 0.

- Complexité :

Le calcul de $x = m^v \bmod n$, tel que $v < n$, nécessite une complexité de $O(log(n))^3$, car on dispose de modulo exponentielle, pour calculer la complexité de l'opération $(y = m \cdot S_a^e) \bmod n$ qui est égale au produit des deux complexités $O(log(n))^2$ et $O(log(n))^3$.
Pour le déchiffrement on a besoin de calculer la complexité du produit $J_a^e \cdot y^v$ qui nous donne la complexité $O(log(n))^2$.
Pour les exposants $v$ et $s$, leur taille dépendra de $\varphi(n)$, et soit celle de n puisque $\varphi(n) = (p-1) \cdot (q-1)$.
Nous concluons que le chiffrement et le déchiffrement de Guillou Guisquater coûtent $O(log(n))^3$ multiplications modulo n.

### 2.2.5 Paillier

Le cryptosystème de Paillier est un cryptosystème basé sur un algorithme asymétrique conçu par Pascal Paillier en 1999. Ce cryptosystème est basé sur un homomorphisme additif. En d'autres termes, avec uniquement la clé publique et le chiffrement de $m_1$ et $m_2$, il est possible de calculer le chiffrement de $m_1 + m_2$ [1, 2, 3, 8, 9].

- Génération de clés :
· Choisir aléatoirement deux nombres premiers de grande taille, indépendants : p et q.
· Calculer la clé publique $n = q \cdot p$ et la clé privée $\varphi(n) = (p-1) \cdot (q-1)$.
· Soit $r$, un entier aléatoire tel que : $0 < r < n$.

· Calculer $r^{-1}$ tel que $r \cdot r^{-1} \bmod n = 1$.

- Chiffrement :

· Soit m un message à chiffrer tel que : $0 < m < n$

· Le message chiffré est alors : $C = (1+n)^m \cdot r^n \bmod n^2$.

- Déchiffrement :

$$m = \frac{(C \cdot (r^{-1})^n \bmod n^2) - 1}{n}$$

- Complexité :

La complexité de chiffrement dépend de la complexité de l'exponentiation modulaire de la formule suivante :
$(1+n)^m \cdot r^n \bmod n^2$, qui est égale à $O(log(n))^3$.
Par contre, la complexité de déchiffrement est égale à la complexité de la formule suivante : $\frac{C \cdot r^{-n} \bmod n^2 - 1}{n}$, qui est égale à $O(log(n))^3$.

## 2.3 Conclusion

Dans ce chapitre nous avons abordé le chiffrement à clé publique, nous nous sommes attardés sur les algorithmes de chiffrement asymétriques basé sur le problème le factorisation, dont l'implémentation fera l'objet du chapitre suivant.

# CHAPITRE 3

# CONCEPTION ET RÉALISATION

## 3.1 Introduction

Ce chapitre a pour objectif l'implémentation des algorithmes de chiffrement vu dans le chapitre précédent. Pour ce faire une bonne conception nous permettra de découper le programme en plusieurs modules et décrira leurs fonctionnalités, ce qui va nous permettre de mieux comprendre le fonctionnement du programme que l'on va réaliser.

Dans un premier temps on présentera le diagramme de classe correspondant à notre projet et les éléments qui le composent. Ensuite, on va expliciter les programmes correspondants à chaque algorithmes.

## 3.2 Conception

Le diagramme de classes est le point essentiel dans un développement orienté objet. En analyse, il a pour but de décrire la structure des entités manipulées par les utilisateurs. En conception, on représente la structure d'un code orienté objet, ou à un niveau de détail plus important, les modules du langage de développement [4, 5, 6].

- Diagramme de classes

Voici le diagramme de classes relatif à l'implémentation des algorithmes de chiffrement que l'on va implémenter :

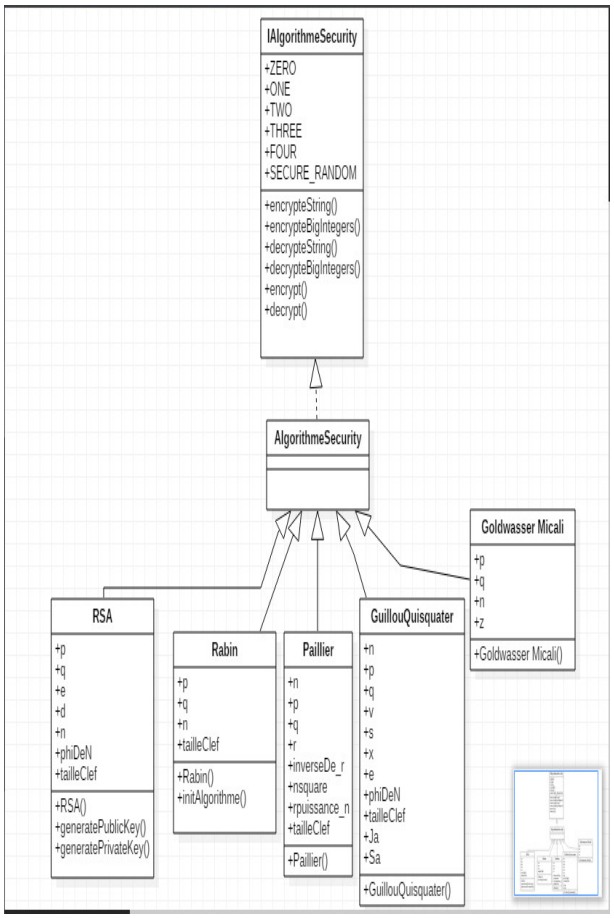



FIGURE 3.1 – Diagramme de classe

## 3.3 Implémentation

### 3.3.1 Outil et langage de programmation utilisés

– **IDE Eclipse**

Eclipse est un IDE, Integrated Development Environment (EDI environnement de développement intégré en français), c'est-à-dire un logiciel qui simplifie la programmation en proposant un certain nombre de raccourcis et d'aide à la programmation. Il est développé par IBM, est gratuit et disponible pour la plupart des systèmes d'exploitation .

Au fur et à mesure qu'on programmez, eclipse compile automatiquement le code qu'on écrive, en soulignant en rouge ou jaune les problème qu'il décèle. Il souligne en rouge les parties du programme qui ne compilent pas, et en jaune les parties qui compilent mais peuvent éventuellement poser problème (on dit qu'eclipse lève un avertissement, ou warning en anglais). Pendant l'écriture du code, cela peut sembler un peu déroutant

au début, puisque tant que la ligne de code n'est pas terminée (en gros jusqu'au point-virgule), eclipse indique une erreur dans le code. Il est déconseillé de continuer d'écrire le programme quand il contient des erreurs, car eclipse est dans ce cas moins performant pour nous aider à écrire le programme [7].

– **JAVA**

Java est le nom d'une technologie mise au point par Sun Microsystems qui permet de produire des logiciels indépendants de toute architecture matérielle. Java est à la fois un langage de programmation et une plateforme d'exécution. Le langage Java a la particularité principale d'être portable sur plusieurs systèmes d'exploitation tels que Windows, Mac OS ou Linux. C'est la plateforme qui garantit la portabilité des applications développées en Java.
Il permet de créer des applications autonomes et de doter les documents html de nouvelles fonctionnalités : animations interactives, applications intégrées, modèles 3D, etc. Ce langage est orienté objet et comprend des éléments spécialement conçus pour la création d'applications multimédia [7].

### 3.3.2 Étude de l'implémentation du projet

Les algorithmes de chiffrement travaillent avec de très grands nombres (BigIntegers). Il conviendra donc d'utiliser la classe **BigInteger** du package **java.math** pour les représenter.

La classe BigInteger permet de représenter des entiers ou des nombres flottants sans les limitations de taille. Elle propose des constantes telles que **ONE** ou **ZERO**, ainsi que des opérations arithmétiques sur des BigIntegers : **add**, **subtract**, **multiply**, etc. Et des méthodes qui permettent de les manipuler telles que :

- *probablePrime*, qui permet de générer un grand nombre qui a une forte probabilité d'être premier.
- *modInverse*, qui permet de calculer l'inverse modulaire d'un nombre (i.e., étant donné e et n, trouver d tel que $e \cdot d \equiv 1 (\mathrm{mod} n)$.
- *modPow*, qui permet de calculer l'élévation à la puissance modulaire ($a^b (\mathrm{mod} n)$).
- *gcd*, qui permet de calculer le pgdc de deux grands nombres.
- . . .

Le projet est intitulé *AlgorithmesSecurite*, il contient les packages suivants :

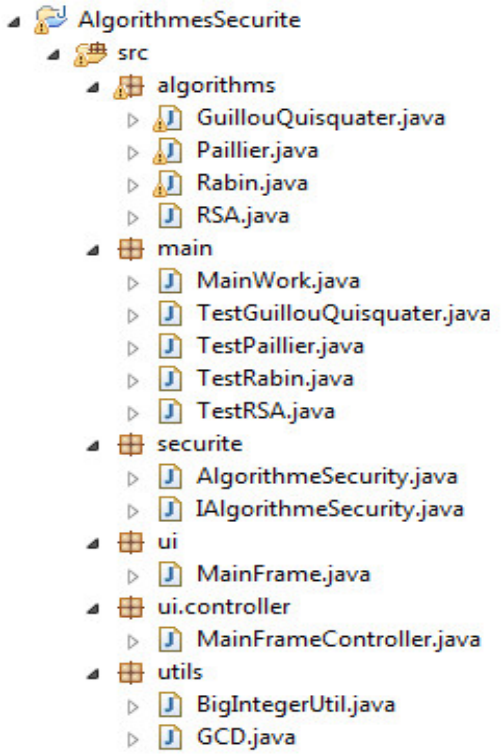



FIGURE 3.2 – Les packages et les classes du projet

On a utilisé six packages :

- *algorithms* :Chaque classe représente un algorithme de chiffrement.
- *main* : Les classes comprises dans celui-ci permettent de lancer l'exécution des algorithmes qui leurs correspondent, par exemple La classe *TestRSA* permet d'exécuter le programme contenu dans la classe *RSA* du package *algorithms*.
- *securite* : contient l'interface *IAlgorithmeSecurity* et la classe *AlgorithmeSecurity* qui implémente ses méthodes.
- *ui* : appelé ui par rapport à User Interface (interface utilisateur en français), il comprend une seule classe nomée "MainFrame", celle guère tout ce qui concerne les composants de l'interface graphique (leurs disposition, taille, couleur,...).
- *ui.controller* : contient la classe "MainFrameController" qui gère les événements qui surviennent sur l'interface graphique.
- *utils* : Ce package contient des classes utilitaires, celles-ci son nom l'indique, ne répond pas directement à un besoin fonctionnel mais plutôt propose des services techniques.

▶ RSA

∗ Programme de génération de clés

Pour crypter avec RSA on choisit d'abord deux nombres premiers, nous allons donc

les générer d'une façon aléatoire par le biais de la méthode *probablePrime* :

```
private void generatePublicKey(){

    if (publicKey == null){
        do{
            publicKey = BigInteger.probablePrime(tailleClef / 2, SECURE_RANDOM);
            e = publicKey;
        }
        while (publicKey.gcd(phiDeN).intValue() != 1 || publicKey.compareTo(n) != -1

                || publicKey.compareTo(p.max(q)) == -1);
    }
}

private void generatePrivateKey(){

    this.generatePublicKey();

    this.d = this.e.modInverse(this.phiDeN);

    this.privateKey = this.d;
}
```

FIGURE 3.3 – Code de génération des clés : privée et publique.

Le calcul de n et de $\varphi(n)$ se fait par :

```
// n
n = p.multiply(q);

phiDeN = (p.subtract(ONE)).multiply(q.subtract(ONE));
```

FIGURE 3.4 – Calcul de n et $\varphi(n)$.

où : **multipy** est une méthode qui permet de calculer la multiplication entre deux BigIntegers, et **subtract** la différence entre ces deux derniers.

∗ Programme de chiffrement (encrypt) et de déchiffrement (decrypt)

```
//chiffrement avec la clef publique
public BigInteger encrypt(BigInteger message) {
    return message.modPow(e, n);
}
```

```
// Decrypte le message avec la clef privee
public BigInteger[] decrypt(BigInteger encrypted) {
    BigInteger[] decryp = { encrypted.modPow(d, n) };
    return decryp;
}
```

FIGURE 3.5 – code de chiffrement et de déchiffrement correspondant à RSA.

▶ Rabin

∗ Programme de génération de clés

Pour crypter avec Rabin on choisit d'abord deux nombres premiers,pour pouvoir calculer n, pour cela on a défini la méthode *initAlgorithme()* qui permet de les générer d'une façon aléatoire :

```
private void initAlgorithme(){
    do{
        p = BigInteger.probablePrime(this.tailleClef / 2, SECURE_RANDOM);
    }while (! p.mod(FOUR).equals(THREE));

        do{
        q = BigInteger.probablePrime(this.tailleClef / 2, SECURE_RANDOM);
    }while (! q.mod(FOUR).equals(THREE) || (q.compareTo(p) == 0));

    n = p.multiply(q);
}
```

FIGURE 3.6 – Code de génération de P, Q et calcul de n.

∗ Programme de chiffrement (encrypt) et de déchiffrement (decrypt)

```
public BigInteger encrypt(BigInteger message) {
    return message.modPow(TWO, n);
}
```

FIGURE 3.7 – code de chiffrement correspondant à Rabin.

```
public BigInteger[] decrypt(BigInteger encrypted, BigInteger origine) {

    BigInteger mp = encrypted.modPow((p.add(ONE)).divide(FOUR), p);
    BigInteger mq = encrypted.modPow((q.add(ONE)).divide(FOUR), q);

    BigInteger [] CoeffBezout = GCD.bigExtendedEuclid(p, q);

    BigInteger Yp = CoeffBezout[1] ;
    BigInteger Yq = CoeffBezout[2];

    // r
    BigInteger r = (Yp.multiply(p).multiply(mq).add(Yq.multiply(q).multiply(mp)));
    if (r.compareTo(ZERO) < 0 || r.compareTo(n) >= 0){
        BigInteger qotNegAddOne = r.divide(n).negate().add(ONE);
        r = r.add( qotNegAddOne.multiply(n) );
    }

    // -r
    BigInteger r_neg = n.subtract(r);
    if (r_neg.compareTo(ZERO) < 0 || r_neg.compareTo(n) >= 0){
        BigInteger qotNegAddOne = r_neg.divide(n).negate().add(ONE);
        r_neg = r_neg.add( qotNegAddOne.multiply(n) );
    }

    // s
    BigInteger s = (Yp.multiply(p).multiply(mq).subtract((Yq.multiply(q).multiply(mp))));
    if (s.compareTo(BigInteger.ZERO) < 0 || s.compareTo(n) >= 0){
        BigInteger qotNegAddOne = s.divide(n).negate().add(ONE);
        s = s.add( qotNegAddOne.multiply(n) );
    }
    // -s
    BigInteger s_neg = n.subtract(s);
    if (s_neg.compareTo(BigInteger.ZERO) < 0 || s_neg.compareTo(n) >= 0){
        BigInteger qotNegAddOne = s_neg.divide(n).negate().add(ONE);
        s_neg = s_neg.add( qotNegAddOne.multiply(n) );
    }


    return resultat;
}
```

FIGURE 3.8 – code de déchiffrement correspondant à Rabin.

▶ Paillier

∗ Programme de génération de clés

```
public Paillier(int tailleClef) {

    this.tailleClef = tailleClef;

    p = new BigInteger(this.tailleClef / 2,3, SECURE_RANDOM);

    do {
        q = new BigInteger(this.tailleClef / 2,3, SECURE_RANDOM);
    } while (q.compareTo(p) == 0);

    // n
    n = p.multiply(q);

    // r

    r = new BigInteger(n.bitLength(), SECURE_RANDOM);
    while( r.compareTo(n) >= 0 || r.compareTo(ZERO) <= 0) {
        r = new BigInteger(n.bitLength(), SECURE_RANDOM);
    }

    inverseDe_r = r.modInverse(n);

    nsquare = n.multiply(n);

    System.out.println (n.intValue());

    if(n.intValue()<0){
        rpuissance_n = ONE.divide(r.pow(n.intValue()));
        System.out.println("n inf to zero = "+n);

        }else
            rpuissance_n = r.pow(n.intValue());
        }
```

FIGURE 3.9 – Code de génération de clés selon Paillier.

∗ Programme de chiffrement (encrypt) et de déchiffrement (decrypt)

```
public BigInteger encrypt(BigInteger message) {
    return ((ONE.add(n)).pow(message.intValue()))
            .multiply(rpuissance_n).mod(nsquare);
}

public BigInteger[] decrypt(BigInteger encrypted) {

BigInteger[] decryp = {((encrypted.multiply(inverseDe_r.pow(n.intValue())))
            .mod(nsquare)).subtract(ONE)).divide(n)};

    return decryp;
}
```

FIGURE 3.10 – code de chiffrement et de déchiffrement correspondant à Paillier.

## 3.4 Conclusion

Ce chapitre nous a permis d'avoir une vision sur la conception de notre application, et sur le code java utilisé pour leur implémentation. Le chapitre suivant quant à lui, sera consacré pour une démonstration de notre application.

# CHAPITRE 4

# MANUEL D'UTILISATION

## 4.1 Introduction

Pour simplifier l'utilisation des programmes de chiffrement vu dans le chapitre précédent, on a créer une interface pour interagir avec l'utilisateur. Le présent chapitre a pour but de nous guider dans l'utilisation de notre application qu'est un outil de "chiffrement et de déchiffrement". Pour l'illustrer, nous avons utilisé des "captures d'écran" permettant de mieux visualiser et suivre pas à pas la procédure de chiffrement et de déchiffrement. On présentera tout d'abord l'interface dédiée à l'utilisateur, puis décrivons les étapes à suivre pour chiffrer ou déchiffrer un message ainsi que quelques exceptions qui pourront survenir lors de la manipulation de cette outil. Enfin on finira par une démonstration de l'exécution de l'algorithme RSA.

La figure sous-djascente représente l'interface dédiée à l'utilisateur :

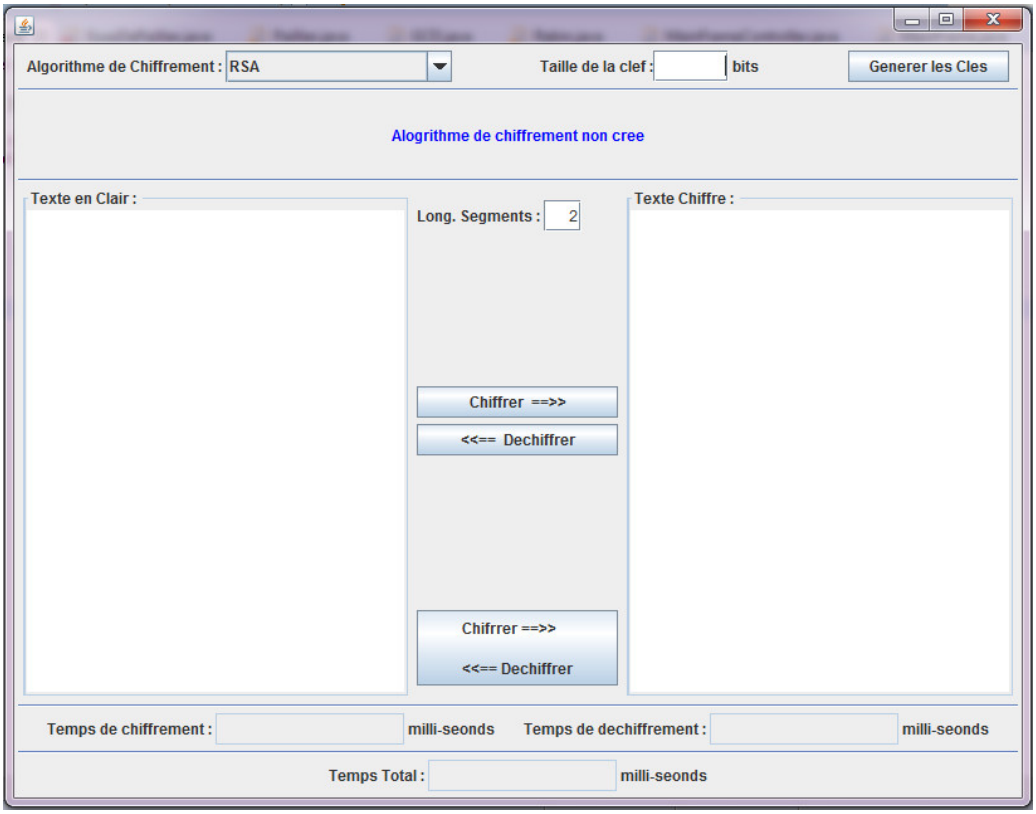



FIGURE 4.1 – L'interface utiliateur.

## 4.2 Composants de l'interface utilisateur et leurs fonctionnalités

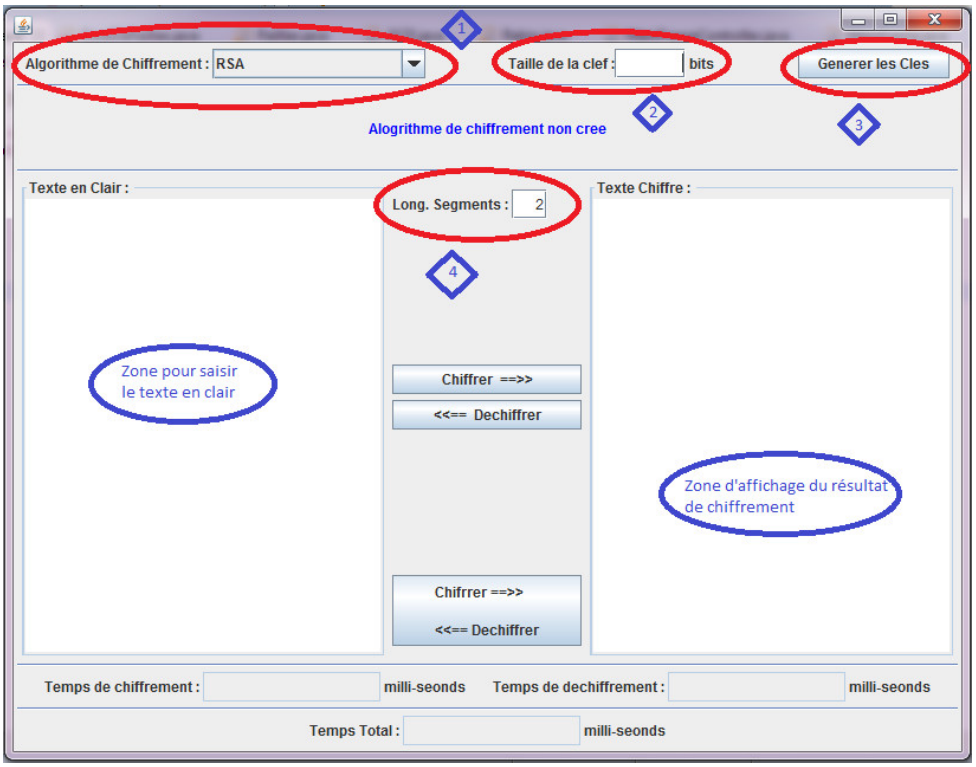



FIGURE 4.2 – Les composants de l'interface utilisateur.

- La composante <1> : *Algorithme de chiffrement*, nous permet de choisir l'algorithme de chiffrement que nous voulons utiliser.
- la composante <2> : *Taille de la clef*, comme son nom l'indique, définit la longueur de la clé en nombre de bits.
- la composante <3> : *Generer les cles*, appel les méthodes, correspondantes à l'algorithme de chiffrement sélectionné, qui permettent de créer la clé publique et la clé privée.
- la composante <4> : *Long. Segment*, spécifie le nombre de caractères que l'on vas considérer comme un bloc. De ce fait, le texte introduit par l'utilisateur sera décomposé en blocs (ou segments) dont la longueur est égales à la valeurs saisie dans la zone de "Long. Segment", pour ensuite être chiffré bloc par bloc.
- la composante : *Texte en clair*, accorde à l'utilisateur une zone de texte pour saisir son texte en clair.
- la composante : *Texte Chiffre*, dans cette zone, on affiche le résultat du chiffrement.
- Le bouton **Chiffrer** récupère le texte en clair saisi, appel les méthodes de chiffrement, qui correspondent à l'algorithme sélectionné, pour le chiffrer et affiche le

résultat dans la zone "Texte Chiffre".

- Le bouton **Dechiffrer** reprend le résultat de chiffrement et par la suite appel les méthodes de déchiffrement pour le déchiffrer et affiche le résultat dans la zone "Texte Clair".
- le bouton "Chiffrer Dechiffrer" effectue l'opération de chiffrement et de déchiffrement à la fois.
- la partie inférieur de la fenêtre, quant à elle, nous retourne les temps d'exécution pour : le chiffrement, déchiffrement, et chiffrement et déchiffrement à la fois (qui est la somme des deux temps d'exécution précédents).

## 4.3 Illustration de l'exécution de RSA

- Étape 01 :

Pour pouvoir chiffrer avec l'algorithme RSA, il faut tout d'abord choisir "RSA" dans la liste déroulante de la boite combinée (combo-box en anglais), ensuite saisir la taille de la clé qu'on souhaite utiliser, dans notre exemple on a mis la taille égale à 512 bits, et enfin appuyer sur le bouton "Generer Les cles" pour produire les clés de chiffrement (public) et de déchiffrement (privee), qui seront affichées juste après.

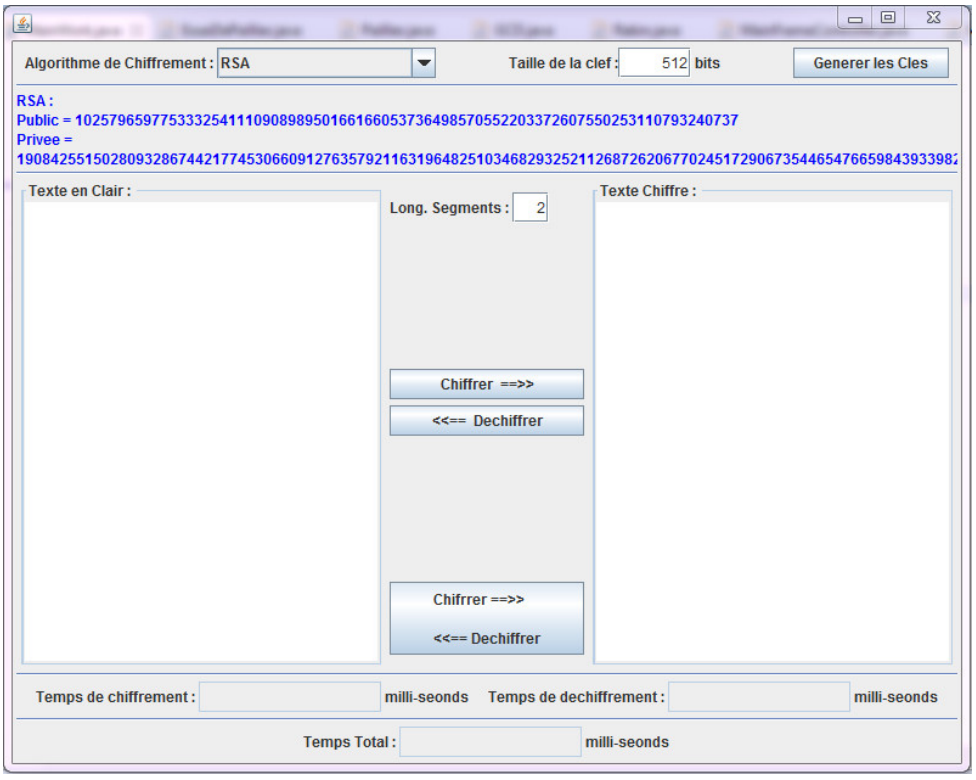



FIGURE 4.3 – Génération de clés pour l'algorithme RSA.

- Étape 02 :

L'utilisateur saisit ce qu'il souhaite chiffrer dans la zone de texte clair, puis appuie sur le bouton chiffrer, le résultat sera affiché dans la zone à coté :

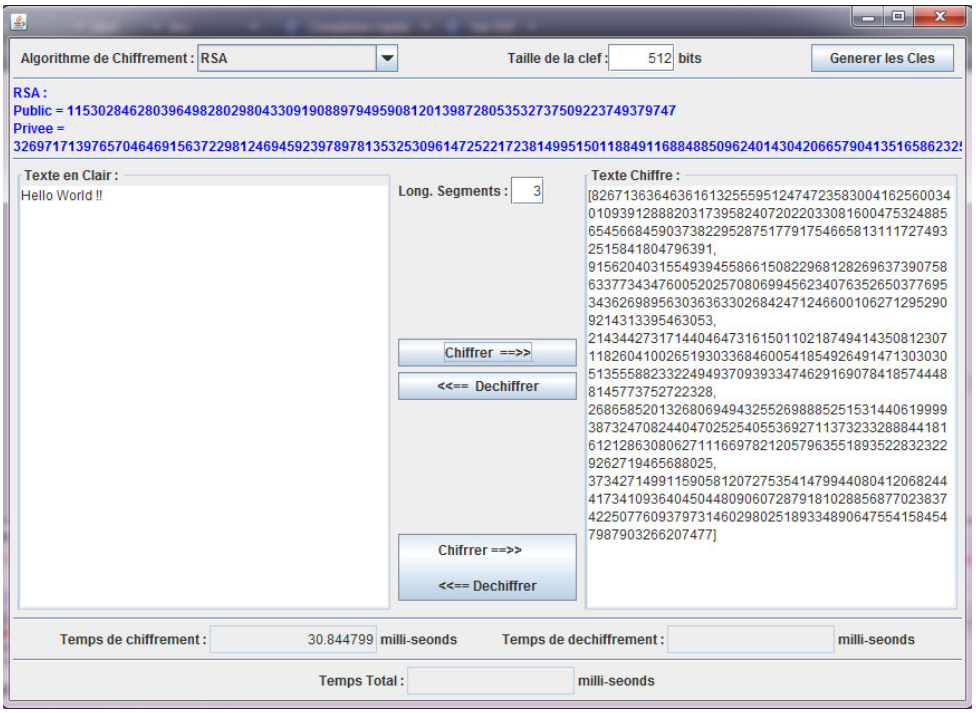



FIGURE 4.4 – Exemple de chiffrement.

Une fois le chiffrement est fait, le temps d'exécution est affiché, dans notre cas il est égale à 30.84479 milli-seconds.

· **Remarques :**

– Dans le cas où l'utilisateur oublie de préciser la taille de la clé, un message sera affiché pour lui demander de l'introduire :

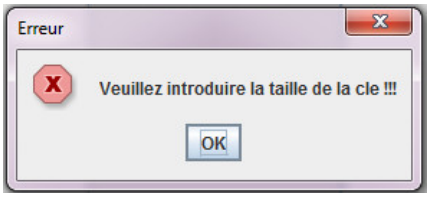



FIGURE 4.5 – Rappel pour l'utilisateur d'introduire la taille de la clé.

– Dans le cas où l'utilisateur n'a pas générer les clés, on affiche le message suivant :

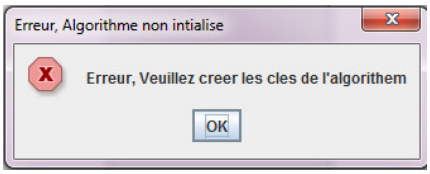



FIGURE 4.6 – Rappel pour l'utilisateur pour générer les clés.

– Dans le cas où l'utilisateur n'introduit pas la longueur du segment, le message suivant est affiché :

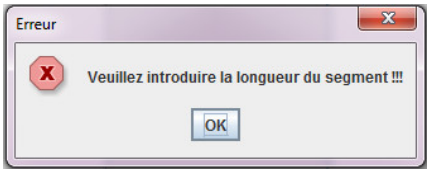



FIGURE 4.7 – Rappel pour l'utilisateur d'introduire la longueur du segment.

- Étape 03 :

Pour déchiffrer, il suffit d'appuyer sur le bouton déchiffrer, le résultat sera affiché dans la zone du texte en clair :

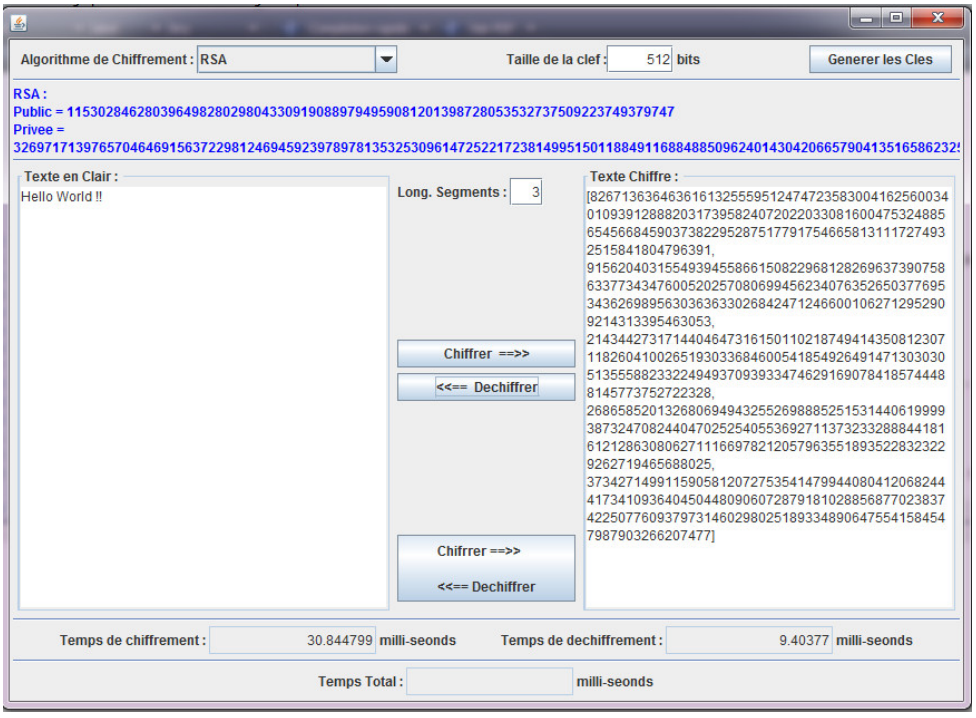



FIGURE 4.8 – Résultat de déchiffrement.

Après avoir déchiffrer, le temps d'exécution va être affiché, dans notre cas il est égale à 9.40377 milli-seconds.

Nous avons une autre façon pour le chiffrement et le déchiffrement grâce au bouton "Chiffrer Dechiffrer", ce dernier lance le chiffrement ensuite le déchiffrement automatiquement et nous retourne les temps : de chiffrement, déchiffrement et le temps total de calcul de l'algorithme :

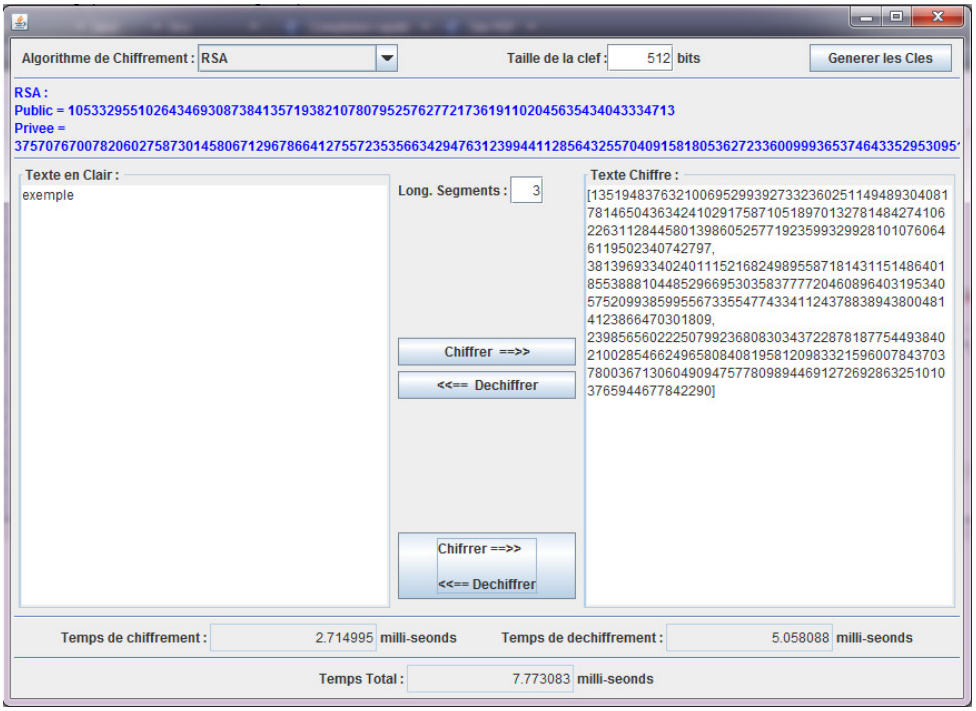



FIGURE 4.9 – Utilisation du bouton "Chiffrer Dechiffrer" pour le calcul.

## 4.4 Conclusion

À travers ce chapitre nous avons vu une description des étapes de l'utilisation de notre outil de chiffrement et de déchiffrement, grâce aux captures d'écran qui visualisent l'interface graphique et ses composants, que l'utilisateur devrai suivre, afin de pouvoir chiffrer et déchiffrer correctement.

# CONCLUSION GÉNÉRALE

Ce mémoire de fin de cycle Master a eu pour objectif l'implémentation et l'analyse des performances des systèmes de chiffrement asymétrique. Pour conclure on vas discuter chaque chapitre de ce rapport.

Pour pouvoir implémenter un algorithme il faut d'abord comprendre son contenu et avoir une idée sur son domaine d'utilisation, pour cela la connaissance du jargon cryptographique des algorithmes de chiffrement que nous sommes sensés implémenter nous a été indispensable, ceci nous a permis de maitriser quelques notions fondamentales de la cryptographie qui nous ont beaucoup aidé dans la suite du travail.

En partant de l'information : " RSA, Rabin, Paillier, Guillou Quisquater, Goldwasser Micali sont des algorithmes de chiffrement asymétrique basés sur le problème de factorisation " on a pu concevoir un diagramme de classe dont la classe mère est "algorithme de sécurité", celle-ci implémente des méthodes qui sont communes à ses sous-classes (génération de clés, chiffrement, déchiffrement) et les classes filles sont des classes qui portent le nom de ces algorithmes et héritent ses méthodes, chacune d'entre elle la défini selon sa façon de fonctionner. cette conception nous a permit de scinder notre projet en plusieurs modules indépendants (chacun représente un algorithme de chiffrement qui s'exécute indépendamment de l'autre).

L'implémentation, quant à elle, a été faite en utilisant le langage de programmation java. L'utilisation de la classe "BigIntegr" a été crucial, vu ses méthodes qui permettent de manipuler les grands chiffres.

L'utilisation des grands nombres (bigintegers) a suscité beaucoup de difficultés lors de l'implémentation. En effet, cette classe de nombres comprend des centaines de chiffres

voir des milliers, ce qui rend l'exécution du programme très lente : celui-ci passe beaucoup de temps à effectuer un calcul, particulièrement une exponentiation modulaire.

Il y a eu un décalage entre les ambitions initiales de ce mémoire et les résultats obtenus parce qu'il existait beaucoup de limites.
D'une part il s'est avéré très difficile de comprendre et de maitriser la théorie mathématique sur laquelle se base chaque algorithme. D'autre part, le manque d'expérience en programmation nous a retardeé : l'exécution des programmes correspondant à ces algorithmes ne nous a pas toujours donné le résultat attendu, par conséquent il fallait toujours vérifier l'algorithme et les instructions utilisées.

En guise de perspectives nous envisageons de :
- Implémenter les deux algorithmes de Guillou Quisquater et Golwasser Micalli,
- comparer l'exécution de ces algorithmes pour voir leur adaptabilité aux équipements à faible ressources.

# BIBLIOGRAPHIE


[1] P. Barthélemy, R. Rolland, P. Véron, Cryptographie, Herme-Science, Paris 2005.

[2] J. Pillou, J. Bay, Tout sur la sécurité informatique, Duno, Paris 2013.

[3] J. Buchmann, Introduction à la cryptographie, Dunod, Paris 2006.

[4] N. Ferguson, Cryptographie en pratique, Vuibert, Paris 2004.

[5] D. Gabay, J. Gabay, UML 2 analyse et conception, Dunod, Paris 2008.

[6] I. Mounier, X. Blanc, UML pour les développeurs, Eyrolles.

[7] C. Delannoy, Programmer en java, Eyrolles.

[8] G. Dubertrt, initiation à la cryptographie, Vuibert ; Paris 2012.

[9] A. Azizi, A. Bonnecaze, M. El Marraki, A. Nitaj, R. Rolland, J.Tenna, cryptographie : de l'histoire aux applications, Eermann éditeur des sciences et des arts, Paris 2012.

# ANNEXE

**Théorème de Bezout**
**Algorithme d'Euclide Etendu**

**Théorème 1** (Identité de Bezout)**.** *Soient $a$ et $b$ deux entiers relatifs et $d$ leur PGCD alors il existe deux entiers $u$ et $v$ tels que : $au + bv = d$.*

**Théorème 2** (Résolution d'une équation diophantienne)**.** *Soient $a$, $b$ et $c$ des entiers, et $d$ le PGCD de $a$ et $b$, alors l'équation $au + bv = c$ admet des solutions entières si et seulement si $c$ est un multiple de $d$.*

**Théorème 3** (Théorème de Bezout)**.** *Soient $a$ et $b$ deux entiers relatifs non nuls. $a$ et $b$ sont premiers entre eux si, et seulement si, il existe deux entiers $u$ et $v$ tels que $au+bv = 1$.*

**Proposition 1** (Solutions multiples pour $u$ et $v$)**.** *Il n'a pas solution unique pour $(u, v)$, mais une infinité $(u + kb, v + ka)$ avec $k \in \mathbb{Z}$*

**Théorème 4** (Théorème de Gauss)**.** *Si un nombre $a$ divise un produit de facteurs et si $a$ est premier avec l'un des deux facteurs alors $a$ divise le deuxième facteur.*

**L'algorithme d'Euclide** permet de calculer le P.G.C.D. de deux entiers naturels $a$ et $b$ tels que $a > b$.
Il consiste à réitérer les manipulations suivantes :

* Effectuer la division euclidienne de $a$ par $b$. Soit $r$ le reste.

* Remplacer $a$ par $b$ et $b$ par $r$. On a $b > r$ d'après la définition de la division euclidienne.

Le P.G.C.D. est le dernier reste non nul.

**L'algorithme d'Euclide Étendu** permet de calculer les coefficients $u$ et $v$ de Bezout.

$r_{-2} := b; r_{-1} := a; u_{-2} := 0; u_{-1} := 1; v_{-2} := 1; v_{-1} := adivb$
$k := -1$
Tant que $rk > 0$ faire
    Début
        $k := k + 1$
        $r_{k-2} = q_k r_{k-1} + r_k (avecq_k = r_{k-2} divr_{k-1} etr_k = r_{k-2} modr_{k-1})$
        $u_k := u_{k-2} - q_k u_{k-1}$
        $v_k := v_{k-2} - q_k v_{k-1}$
        On a la relation : $r_k = au_k + bv_k$
    Fin
$u = u_{k-1} q_k$
$v = v_{k-1} q_k$

# Etapes d'installation des outils nécessaires à l'implémentation des algorithmes de chiffrement

## Installation du JDK

### Téléchargement

Il faut commencer par télécharger le JDK, voila un lien ou nous pouvons le télécharger : https ://www.oracle.com/tecknetwork/java/javase/downloads/idex.html.
Ensuite, cliquer sur le bouton "Downloade JDK". Puis cliquer sur le bouton : "Downloade " dans la page qui s'affiche.
Sur la page suivante, choisir le système d'exploitation (Plateforme) puis cliquer sur le bouton "Continuer".
Enfin, il faut cliquer sur l'exécutable à télécharge :jdk-6u21-windows-i568.exe dans mon cas .

### Installation

Pour lancer l'installation, il faut faire un double-clique (avec le bouton gauche de la souris) sur l'exécutable précédemment téléchargé et se laisser guider.

# Installation d'Eclipse

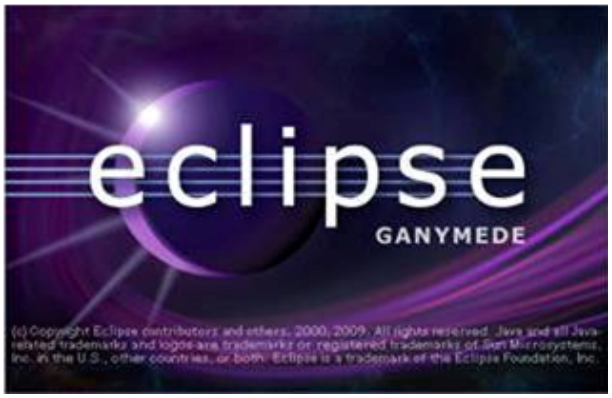
## Téléchargement

Pour commencer nous devons télécharger Eclipse, voila un lien où nous pouvons le faire :
http ://www.objis.com/formation-java/tutoriel-java-installation-eclipse.html .
Nous aurons le chois entre beaucoup de version d'Eclipse mais l'idéal est de choisir " Eclipse IDE for Java Developers " :

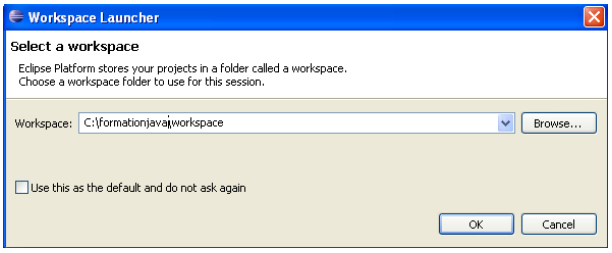
## Installation

L'installation d'Eclipse se résume en réalité à décompresser le fichier téléchargé .nous aurons après à le placer n' importe où sur le disque dur.

Double-clique sur le fichier exécutable eclipse.exe présent dans le répertoire 'eclipse' et Eclipse se lance.




Un écran apparaît ensuite pour nous demander le 'workspace' (Répertoire de travail). C'est dans ce répertoire que Eclipse stockera les programmes (*.java) et fichiers de configuration (*.xml, *.properties, ...) de vos projets.



**Résumé**

La cryptographie est une technologie essentielle dans la sécurité informatique. Ses techniques modernes ont de nombreux usages.Par exemple, elles permettent de garder secrets des messages ou des données enregistrées, signer des documents électroniques, de contrôler des accès, d'assurer la protection du copyright et la confidentialité.

Autant d'usages importants faits que les utilisateurs doivent être capables d'estimer l'efficacité et la sécurité des techniques cryptographiques.

Ce rapport comprend une présentation et une description précise des principaux algorithmes de chiffrement asymétriques basés sur le problème de factorisation, ainsi qu'une implémentation sous java, qui permet de voir concrètement leurs fonctionnement et leur performance.

**Mots clés :** cryptographie, copyright, Confidentialité, Algorithme, Java.

**Abstract**

Cryptography is an essential technology in the computer security. Its modern technologies have many uses. For example, they make it possible to keep secret messages or recorded data, to sign electronic documents, to control accesses, to ensure the protection of the copyright and the confidentiality.

As many made important uses than the users must be able to estimate the effectiveness and the security of the cryptographic techniques.

This report contains a presentation and a precise description of the principal asymmetrical encryption algorithms based on the problem of factorization, as well as an implementation under java, which makes it possible to concretely see their operation and their performance.

**KeyWords :** cryptography, copyright, Privacy, Algorithm, Java .