TABLE DES MATIERES

REMERCIEMENTS i
TABLE DES MATIERES ii
NOTATIONS v
INTRODUCTION 1
CHAPITRE 1 L'ART DE LA STEGANOGRAPHIE 3
1.1 Introduction 3
1.2 Définitions de la stéganographie 3
1.3 Historique 4
1.3.1 Une technique antique 4
1.3.2 Du moyen-âge à l'avant-guerre 5
1.3.3 Des 2 guerres mondiales à aujourd'hui 5
1.4 Alice, Bob et Wendy 6
1.5 Architectures stéganographiques 8
1.5.1 Stéganographie pure 9
1.5.2 Stéganographie à clé secrète 9
1.5.3 Stéganographie à clé publique 10
1.6 Caractéristiques de Schéma Stéganographique 11
1.7 La Stéganalyse 11
1.7.1 Définition 11
1.7.2 Sécurité parfaite 12
1.7.3 Détection des stégo-objets 13
1.7.4 Techniques de stéganalyse 13
1.8 Tatouage numérique ou Watermarking 14
1.8.1 Technologie du watermarking 14
1.8.2 Qualités d'un tatouage 16
1.8.3 Visibilité 18
1.8.4 Robustesse et fragilité 19
1.8.5 Applications 21
1.9 Comparaison de la stéganographie, du tatouage et du fingerprinting 22
1.10 Conclusion 22
CHAPITRE 2 ALGORITHMES DE STEGANOGRAPHIE ET DE CRYPTAGE UTILISES 23
2.1 Introduction 23
2.2 Techniques stéganographiques modernes 23
2.2.1 LSB (Least Significant Bit) 24
2.2.2 L'algorithme F5 25
2.3 Cryptographie 33
2.3.1 Chiffrement à clé privée 33
2.3.2 Chiffrement à clé publique 34
2.3.3 L'advanced Encryption Standard ou « AES » 34
2.4 Conclusion 42

CHAPITRE 3 SYSTEMES D'EXPLOITATION POUR MOBILE 43
3.1 Introduction 43
3.2 Les smartphones 43
3.2.1 iOS 45
3.2.2 Windows Phone 49
3.2.3 BlackBerry OS 52
3.2.4 Android 54
3.3 Parts de marché 59
3.4 Conclusion 63
CHAPITRE 4 CONCEPTION ET DEVELOPPEMENT DE L'APPLICATION 64
4.1 Introduction 64
4.2 Les difficultés du développement pour des systèmes embarqués 64
4.3 Outils de développement d'Android 65
4.3.1 Le Java Development Kit 66
4.3.2 Eclipse, l'ADT et le SDK d'Android 66
4.3.3 Configuration de l'émulateur de téléphone AVD et du terminal réel 68
4.4 Contenu d'un programme Android 69
4.5 Cycle de vie d'une activité 70
4.6 Analyse et Conception 71
4.6.1 Identification des besoins fonctionnels 72
4.6.2 Identification des besoins non fonctionnels 72
4.6.3 Diagramme de cas d'utilisation 73
4.6.4 Diagramme d'activité 78
4.6.5 Techniques utilisées 79
4.7 Implémentation 82
4.7.1 Spécifications du projet 82
4.7.2 Paramètres d'entrées et de sortie de l'application 82
4.7.3 Diagramme de classe de l'application 83
4.7.4 Interfaces graphiques de l'application 84
4.8 Conclusion 85
CHAPITRE 5 EVALUATIONS ET TESTS DE L'APPLICATION 86
5.1 Introduction 86
5.2 Evaluation et test 86
5.2.1 Le PSNR 88
5.2.2 Tests, résultats et interprétations 89
5.3 Bilan sur les deux algorithmes de l'application 101
5.4 Conclusion 101
CONCLUSION 102
ANNEXE 1 TECHNIQUE DE MATRIX EMBEDDING 104
ANNEXE 2 MARQUEURS JPEG COURANTS 110
ANNEXE 3 PROCEDURE SUBBYTES VIA S-BOX DU CHIFFREMENT AES 111

ANNEXE 4 COMPLEMENTS SUR LE DEVELOPPEMENT DE L'APPLICATION 112
BIBLIOGRAPHIE 114
FICHE DE RENSEIGNEMENT 116
RESUME 1
ABSTRACT 1

NOTATIONS

1. Minuscules latines

c	: Couverture en stéganographie
f	: Fonction de détection des stégo-objets
k	: Clé secrète
m	: Message secret à transmettre
n	: Longueur du mot codé en codage linéaire
n_r	: Nombre de tours dans le chiffrement AES
s	: Syndrome en codage linéaire
s(.)	: Transformation SubBytes dans le chiffrement AES
t_d	: Tableau de données dans le cryptage AES
t_k	: Tableau de clés dans le cryptage AES

2. Majuscules latines

AC	: Coefficient DCT, autre que le coefficient $S_{0,0}$
B	: Valeur de la couleur bleu dans le système RGB
C	: Ensemble des possibilités de couverture ou Capacité
Cb	: Valeur de la chrominance bleue dans le système YCbCr
Cr	: Valeur de la chrominance rouge dans le système YCbCr
D	: Dynamique d'une image
D(.,.)	: Fonction d'extraction pour un système de stéganographie pure
D(.||.)	: Entropie relative entre deux distributions
D_K(.,.)	: Fonction d'extraction de stéganographie à clé secrète
DC	: Coefficient DCT $S_{0,0}$
E(.,.)	: Fonction d'insertion pour un système de stéganographie pure
E_K(.,.,.)	: Fonction d'insertion de stéganographie à clé secrète
F5	: Algorithme de stéganographie F5
G	: Couleur verte dans le système RGB ou matrice génératrice
G{.,.,.,.}	: Système de stéganographie

H : Matrice de contrôle de parité en codage linéaire

I : Image hôte

I’ : Image marquée

K : Ensemble des clés secrètes

M : Ensemble des messages secrets

N_k : Nombre de colonnes du tableau de clés t_k dans le cryptage AES

P : Précision du système YCbCr

P_C : Probabilité de distribution de la variable aléatoire C

P_S : Probabilité de distribution des stégo-objets

Q : Facteur de qualité dans la compression JPEG

R : Valeur de la couleur rouge dans le système RGB

RL : Nombre de 0 précédant le coefficient à coder dans le codage RLE

S : Nombre de bits pour coder le coefficient dans le codage RLE

St : Texte en clair dans le chiffrement AES

S_{uv} : Coefficient DCT de coordonnées (u, v)

S_{xy} : Pixel de coordonnées (x, y)

TK : Clé de tour dans le chiffrement AES

V : Valeur du coefficient codée sur S bits dans le codage RLE

W : Tableau étendu dans AES

Y : Information de luminance dans le système YCbCr

3. Minuscules grecques

α : Elément de F_{256} ou taille relative du message en matrix embedding

β : Probabilité qu'un attaquant passif fasse des erreurs de type II

δ : Distance minimale de Hamming

$\in$: Critère de sécurité d’un système de stéganographie $\in$-securisé

π : Constante pi

4. Abréviations

ADT	: Android Development Tools
AES	: Advanced Encryption Standard
API	: Application Programing Interface
AVD	: Android Virtual Device
BMP	: Bitmap
dB	: Décibel
DCT	: Discrete Cosine Transform
DES	: Data Encryption Standard
DWT	: Discrete Wavelet Transform
GHz	: Gigahertz
Go	: Gigaoctet
HTML	: Hypertext Markup Language
HTTP	: Hypertext Transfer Protocol
IDC	: International Data Corporation
IDE	: Integrated Development Environment
iOS	: iPhone OS
IP	: Internet Protocol
ISO	: International Standardization Organization
ITU	: International Telecommunication Union
JDK	: Java Development Kit
JPEG	: Joint Photographic Experts Group
JRE	: Java Runtime Environment
JVM	: Java virtual machine
Ko	: Kilooctet
LSB	: Least Significant Bit

MMS	: Multimedia Messaging Service
Mo	: Mégaoctet
MSE	: Mean Square Error
OHA	: Open Handset Alliance
OS	: Operating System
PNG	: Portable Network Graphics
PSNR	: Peak Signal-to-Noise Ratio
RAM	: Random Access Memory
RGB	: Red Green Blue
RLE	: Run Length Encoding
SDK	: Software Development Kit
SIP	: Session Initiation Protocol
SMS	: Short Message Service
TCP	: Transmission Control Protocol
Wi-Fi	: Wireless Fidelity
WWDC	: Worldwide Developers Conference
XML	: Extensible Markup Language

5. Notations spéciales

$\mathbb{Q}$	: Ensemble des nombres rationnels
$\mathbb{F}_n$	: Corps fini à n éléments

INTRODUCTION

La technologie ne cesse d'évoluer considérablement ces dernières années et semble ne pas vouloir s'arrêter, ce qui conduit à la banalisation de celle-ci. En effet, la plupart des gens actuellement y ont accès : par exemple, une grande majorité peut s'offrir un téléphone portable doté de capacité de traitement de plus en plus performante, et cela à un prix plus ou moins abordable, contrairement à auparavant.

En outre, ces progrès en termes de technologies apportent un nouveau concept : l'importance de l'information, l'oxygène des temps modernes. L'information peut être une source de richesses et de pouvoirs pour une entreprise, ultraconfidentielle dans le domaine de la communication militaire, vitale ou tout simplement privée pour certaines personnes.

À la lumière de ce constat, donner une protection adéquate à l'information devient un enjeu primordial pour tous les acteurs économiques (entreprises, gouvernements, mais aussi les utilisateurs/consommateurs).

La cryptographie, une première solution, consiste à chiffrer l'information, c'est-à-dire, à appliquer une certaine transformation à celle-ci, de façon à la rendre illisible pour les personnes non concernées. Toutefois, un problème se pose : l'information chiffrée attire trop l'attention. En effet, certes les personnes malintentionnées ne savent pas encore sur le moment comment la déchiffrer, cependant, elles savent que c'est une information chiffrée suspecte, et donc importante. Ces personnes peuvent ensuite essayer de la décrypter en utilisant la cryptanalyse.

C'est là qu'intervient la stéganographie, un tout autre art, une autre science souvent confondue avec la cryptographie. La stéganographie arrive en renforcement au chiffrement de données. Quelle meilleure protection y a-t-il que de faire passer une information sans que les personnes susceptibles de l'intercepter aient même conscience de son existence ?

Ceci étant, le but de ce travail est donc axé sur la conception et le développement d'une application mobile de stéganographie, une technique qui permet de répondre à notre besoin, c'est-à-dire de protéger l'information d'une certaine façon en la dissimulant dans un autre support anodin, d'où le titre de ce mémoire : « Développement d'application mobile de stéganographie ».

Le contenu de ce mémoire est divisé en cinq parties :

- La première partie, consiste à présenter l'art de la stéganographie, à décrire son principe afin de mieux cerner son intérêt.

- La deuxième partie concerne les techniques de stéganographie et de chiffrement qui seront utilisées par l'application.
- La troisième partie, parle plutôt d'une comparaison des meilleurs systèmes d'exploitation mobiles actuels, et explique le choix de la plateforme Android comme plateforme de développement.
- La quatrième partie relate les phases de conception et de développement de l'application, ainsi que de son implémentation.
- La dernière partie est réservée aux différents tests et évaluations de la performance de l'application.

CHAPITRE 1
L'ART DE LA STEGANOGRAPHIE

1.1 Introduction

Dans ce chapitre, nous présentons tout d'abord quelques définitions de la stéganographie, un petit historique des techniques utilisées pour bien cerner la philosophie du domaine, et les concepts d'emploi. Nous posons ensuite les bases de la stéganographie moderne et mettons en évidence les propriétés intrinsèques des schémas de la stéganographie. Une notion de stéganalyse y sera présentée. Nous en déduirons ainsi les services de sécurité offerts par de telles techniques, ainsi que les règles fondamentales de leur mise en œuvre. En outre, nous verrons brièvement une technique dérivée de la stéganographie appelée tatouage numérique, mais pourtant qui diffère de celle-ci par l'utilisation et le but recherché.

1.2 Définitions de la stéganographie

La ***stéganographie*** est ***l'art de la dissimulation de communications***. Contrairement à la cryptographie, la stéganographie n'a pas pour objectif de sécuriser une communication, ***mais d'en cacher l'existence même***. Les deux disciplines ont donc chacune leurs propres domaines de compétence. Dans certaines situations, le fait même de vouloir transmettre des données de manière chiffrées sera jugé comme ***suspect*** [1].

La stéganographie provient étymologiquement de la combinaison des mots grecs ***Stéganô***, signifiant Je ***couvre***, et ***Graphô***, signifiant J'***écris***, soit littéralement traduisible par ***Je couvre ce que j'écris***. Ainsi l'objectif premier de la stéganographie est de dissimuler une information, d'en cacher son existence, là où la cryptographie cherche à rendre illisible cette information.

Pour bien comprendre ce concept, comparons-la avec la cryptographie plus connue. Dans le cas de la cryptographie, un message est chiffré avec une clé de chiffrement qui reste secrète, et garantit le secret du message. Une fois chiffré le message est illisible, ce message chiffré peut circuler librement sans risquer d'être dévoilé, car son intégrité est liée au secret de la clé de chiffrement uniquement connue par les personnes qui sont censées accéder au contenu du message. Dans ce cas de figure les personnes qui voient le message chiffré savent qu'il contient une information, mais ne peuvent la lire. Concernant la stéganographie, c'est l'existence de ce message qui est cachée. C'est-à-dire que le secret du message est préservé par le fait que personne, hormis les personnes auxquelles le message est destiné, ne doit savoir qu'un message est transmis.

Si dans le cas de la cryptographie, la sécurité de l'information est garantie par la clé de chiffrement, dans le cas de la stéganographie ***la sécurité de l'information est garantie par le secret de la manière dont est dissimulé le message***.

En stéganographie on cherche à cacher un message secret (« secret message » en anglais) au sein ***d'un objet de couverture (cover object)***. La compilation du message secret et de l'objet de couverture donnera un ***objet stégo*** ou encore ***stégo-objet (stego object)***, qui devra être d'apparence similaire à l'objet de couverture, et contenir le message secret afin de pouvoir le transmettre sans que personne hormis les personnes pour lesquelles le message secret est destiné, ne suspecte l'existence de ce message secret au sein de l'objet de couverture [2].

1.3 Historique

1.3.1 Une technique antique

La première forme de stéganographie répertoriée nous vient d'une histoire Grecque signée Hérodote et datant du 5ème siècle avant Jésus-Christ. L'auteur nous relate la révolte contre les lois Perses. Les Grecs utilisaient certains esclaves pour transmettre les messages. Ceux-ci étaient écrits sur les crânes des messagers, et passaient donc inaperçus lorsque les cheveux repoussaient. Une fois ses cheveux suffisamment longs, le messager pouvait être envoyé, avec l'ordre de se faire raser le crâne, une fois arrivé à destination. Le principal désavantage de cette méthode était l'attente pour l'envoi d'un message. Une autre technique était d'utiliser des tablettes de cire (cf. figure 1.01). Une fois la cire raclée, on gravait le message dans le bois de la tablette. Il suffisait ensuite d'y remettre de la cire, et le message était parfaitement caché.

Figure 1.01 : *Tablette de cire antique*

En Chine ancienne, les messages étaient écrits sur de la soie, qui était ensuite roulée en boule, elle-même recouverte de cire. Un messager devait enfin avaler cette boule.

Dès le I^{er} siècle avant Jésus-Christ, les romains utilisaient l'encre invisible, qui fut la plus utilisée des méthodes de stéganographie à travers les siècles. On écrit, au milieu des textes écrits à l'encre, un message à l'aide de jus de citron, de lait ou de certains produits chimiques. Il est invisible à l'œil, mais une simple flamme, ou un bain dans un réactif chimique, révèle le message [1].

1.3.2 Du moyen-âge à l'avant-guerre

Au cours du moyen-âge et de la renaissance se développèrent des techniques de stéganographie linguistiques dont la plus connue est l'acrostiche, qui repose sur l'écriture d'un message caché qui sera lu, en lisant la première lettre ou le premier mot d'une ligne de haut en bas et non de gauche à droite. Parmi les plus célèbres acrostiches, on peut citer les correspondances d'Alfred De Musset et Georges Sand. Lewis Carroll en est aussi friand dans ses livres et plus particulièrement dans le livre *A travers le miroir* où il révèle sous forme d'acrostiche le nom de son héroïne Alice [2].

D'autres techniques consistent à cacher un message en jouant sur la ponctuation du texte, l'espacement entre les mots, ou des erreurs volontaires sur le style de l'écriture (typographies, taille de polices de caractères, etc…). On voit aussi apparaître au cours de cette période des techniques utilisant la musique pour envoyer des messages de manière stéganographique. Un scientifique allemand, Gaspart Schott (1608-1666) explique dans son livre « *Schola Steganographica* » comment dissimuler des messages en utilisant des notes de musique. Les notes des partitions de musique correspondent alors à des lettres qui forment le message caché, pouvant ainsi être transmis en récupérant la partition ; ou bien en écoutant les notes jouées par un musicien comme expliqué par John Wilkins en 1694, qui montre par ailleurs comment transmettre un message avec l'utilisation de figures géométriques, chaque figure (ligne, rectangle, carré, etc…) dans sa position correspondant à une lettre de l'alphabet [1].

1.3.3 Des 2 guerres mondiales à aujourd'hui

La période correspondant aux deux guerres mondiales et la guerre froide est en quelque sorte l'apogée de l'utilisation de la stéganographie. Lors de la première guerre, les espions dissimulaient des microfilms contenant des images réduites par réductions successives sous leurs ongles, dans les oreilles, etc….

Les techniques d'encres invisibles refont aussi leurs apparitions de manière améliorée, avec des procédés chimiques qui nécessitent un premier produit chimique pour cacher un message, et un deuxième différent pour le révéler.

Les services secrets se sont aussi servis de la technique des micros points, qui consiste à réduire un message ou une image à une taille microscopique, et à l'insérer dans un signe de ponctuation d'un texte contenu par exemple dans un magazine ou un journal.

Ces dix dernières années ont vu un regain d'intérêt pour l'utilisation de la stéganographie, principalement suite à l'attaque des deux tours du World Trade Center du 11 Septembre 2001.
Les services secrets américains soupçonnent vigoureusement Oussama Ben Laden et ses hommes, d'avoir échangé des informations via l'utilisation d'images pornographiques sur Internet cachant des messages.

Le succès de romans tels que le *Da Vinci Code* de Dan Brown ou la série américaine *Prison Break* n'est pas non plus étranger à cette attention nouvelle.

Par ailleurs, de la stéganographie est née une variante, à savoir le tatouage numérique, qui consiste à inscrire des données sur des supports numériques pour en certifier l'auteur, technique qui intéresse au plus haut point les majors du disque et du cinéma qui cherchent à l'utiliser pour limiter le piratage numérique [2].

1.4 Alice, Bob et Wendy

Pour expliquer le principe et l'utilité de la stéganographie appelée aussi communication invisible, Simmons [3] a été le premier à proposer le problème des prisonniers avec pour acteurs de ce problème : Alice, Bob et Wendy.

Alice et Bob sont deux personnes arrêtées et placées en prison dans deux cellules de prison différentes. Ces deux prisonniers cherchent à s'évader et pour cela elles doivent planifier leur action en s'échangeant des informations (heure et date de la tentative de l'évasion par exemple).

Malheureusement leurs seules communications possibles doivent se faire par l'intermédiaire de Wendy le gardien de prison. Ainsi ils ne peuvent s'échanger des lettres, où sont par exemple écrites clairement des informations qui doivent rester secrètes. Wendy a la possibilité de lire ses lettres et donc de ne pas les transmettre s'il pense détecter une information compromettante. De la même manière si Alice et Bob tentent de crypter les informations pour parer à ce problème,

Wendy considérera comme suspect le message de par son caractère illisible et ne le transmettra pas.

Alice et Bob doivent alors trouver un subterfuge pour que Wendy transmette le message sans y détecter la présence d'une information sur leur future évasion. Pour cela ils doivent créer ce que Simmons appelle un *canal subliminal*, autrement dit un canal secret de communication dont seuls Alice et Bob connaissent l'existence, malgré le fait que Wendy fasse partie intégrante de la transmission de ce message. La figure 1.02 représente la configuration de la transmission de message stéganographique entre Alice, Bob et Wendy [2].

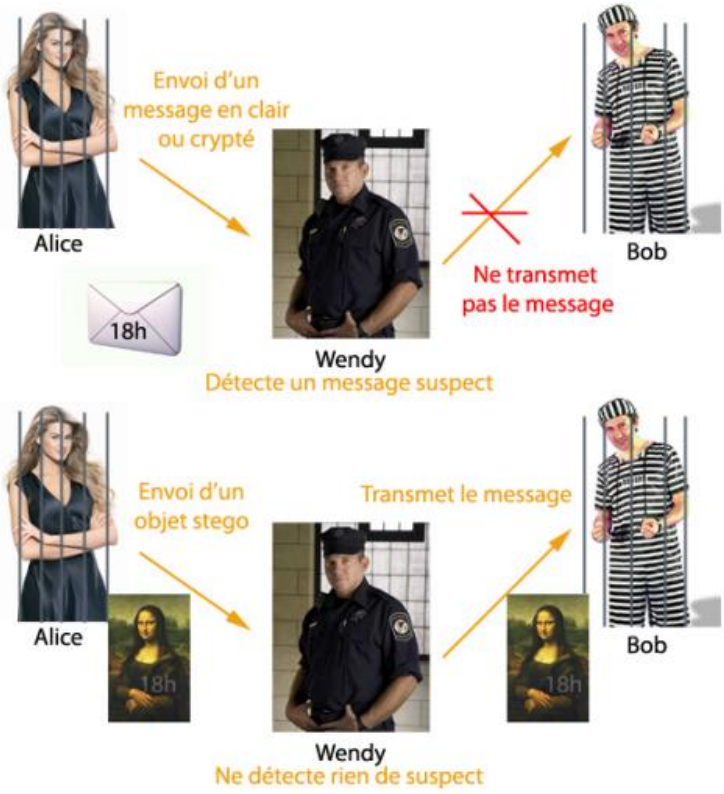


Figure 1.02 : *Le problème des prisonniers*

Pour arriver à créer ce canal subliminal, Alice peut par exemple colorier une image d'une certaine façon, pour que les couleurs correspondent à un code compréhensible par Bob, et qui décrit la date et l'heure de l'évasion. Ainsi si Alice envoie cette image par l'intermédiaire de Wendy, Wendy pensera seulement transmettre une image à Bob, et en aucun cas une information. A la réception de cette image, Bob connaissant la méthode de lecture du message secret, sera alors capable de connaître les informations envoyées par Alice. Ils pourront ensuite continuer à se transmettre des images coloriées pour créer ce canal subliminal.

Bien entendu cette approche nécessite que le moyen de communication, à savoir la technique de stéganographie employée pour créer le canal subliminal, ait été défini avant l'emprisonnement d'Alice et Bob [2].

Le modèle des prisonniers peut facilement être transposé à toute communication entre deux dispositifs qui cherchent à communiquer de manière invisible, et où Wendy serait un attaquant qui écoute le réseau (eavesdropper).

Par ailleurs l'attaquant en stéganographie peut être de trois types :

- **Passif :** On parlera d'un attaquant passif quand l'attaquant ne cherche qu'à détecter une information circulant entre Alice et Bob. Il ne cherche pas forcément à découvrir le message secret mais seulement à découvrir son existence.
- **Actif :** un attaquant actif est un attaquant qui modifie l'objet de couverture, et peut altérer dans certains cas le message secret.
- **Malicieux :** un attaquant malicieux va modifier le message secret contenu dans l'objet de couverture, il peut par exemple essayer de se faire passer pour un des deux prisonniers.

1.5 Architectures stéganographiques

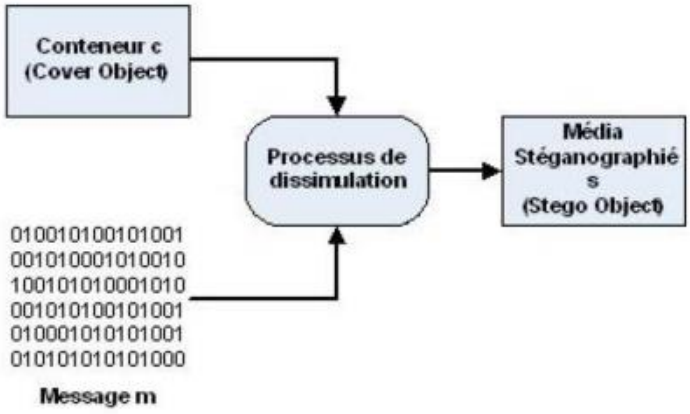


Figure 1.03 : *Schéma simplifié de stéganographie*

Dans une architecture stéganographique, il y a principalement deux éléments. D'un côté ***un processus de dissimulation***, de l'autre ***un processus de recouvrement***. Un processus de dissimulation simplifié peut être donné par le schéma de la figure 1.03 [1].

Il existe trois types d'architecture de stéganographie, correspondant de près à ce qui existe en cryptographie.

1.5.1 Stéganographie pure

La stéganographie pure correspond à une utilisation de la stéganographie, où le secret du message ne correspond qu'à la méthode de dissimulation de ce message et à la méthode de récupération de ce message secret.

Le quadruplet $G = \{ C, M, D, E \}$, où C est l'ensemble des possibilités de couverture, M l'ensemble des messages secrets avec $| C | \geq | M |$, $E : C \times M \rightarrow C$ la fonction d'insertion et $D : C \rightarrow M$ la fonction d'extraction, avec la propriété que $D(E(c, m)) = m$ pour tout $m \in M$ et $c \in C$ est appelé : ***système de stéganographie pure***.

Pour cette méthode, deux correspondants n'ont besoin d'échanger que la méthode de dissimulation et de récupération du message. Cela sous-entend que si un attaquant connaît ces méthodes, il n'y a plus de secret et il peut lire directement le message secret. Selon la méthode de stéganographie utilisée, l'attaquant peut trouver facilement la méthode pour découvrir le message. C'est donc le niveau de sécurité le plus faible pour l'utilisation de la stéganographie [2].

1.5.2 Stéganographie à clé secrète

La stéganographie à clé secrète correspond à la combinaison de l'utilisation d'une méthode de stéganographie pour dissimuler le message, et au cryptage préalable de ce message avant son incrustation dans l'objet de couverture.

Le quintuplé $G = \{ C, M, K, D_K , E_K \}$, où C est l'ensemble des possibilités de couverture, M l'ensemble des messages secrets avec $| C | \geq | M |$, K l'ensemble des clés secrètes, $E_K : C \times M \times K \rightarrow M$ la fonction d'insertion et $D_K : C \times K \rightarrow M$ la fonction d'extraction, avec la propriété que $D_K (E_K(c, m, k), k) = m$ pour tout $m \in M$, $c \in C$ et $k \in K$ est appelé : ***système de stéganographie à clé secrète***.

Pour mettre en place cette technique, il est nécessaire que deux correspondants connaissent d'une part la méthode de dissimulation et de récupération du message secret, et d'autre part qu'ils partagent une clé commune de chiffrement. Ainsi un attaquant qui trouverait la méthode stéganographique pour dissimuler l'information, ne pourrait pas pour autant lire le message secret s'il ne connait pas la clé de chiffrement [2].

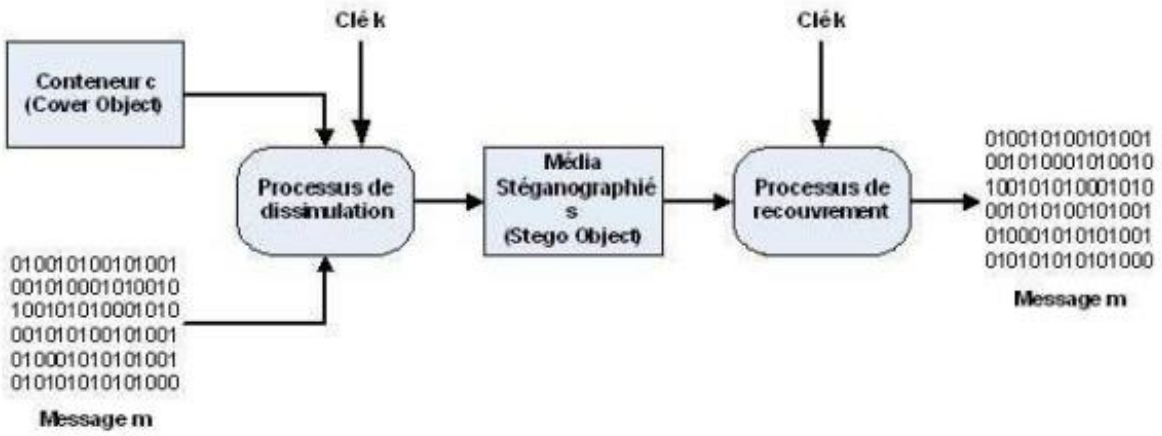


Figure 1.04 : *Schéma de stéganographie à clé secrète*

La sécurité est donc globalement accrue par rapport à l'utilisation d'une technique de stéganographie pure. Cependant l'utilisation de clé symétrique ne permet pas de garantir l'authenticité de l'envoyeur du message, et peut permettre à un attaquant malicieux connaissant la clé de chiffrement de se faire passer pour l'expéditeur du message.

1.5.3 Stéganographie à clé publique

La stéganographie à clé publique reprend la définition de la stéganographie à clé symétrique hormis le fait que le chiffrement ne se fait plus à l'aide d'une clé partagée, mais grâce à l'utilisation de clés, publique et privée.

Ainsi si Alice souhaite envoyer un message secrètement à Bob, elle va préalablement chiffrer ce message avec la clé publique de Bob, et éventuellement le signer avec sa clé privée pour en assurer son authenticité. Puis elle le dissimulera avec une technique de stéganographie dans un objet de couverture, qu'elle fera transmettre à Bob. Celui-ci n'aura plus qu'à récupérer le message chiffré, à le déchiffrer, avec sa clé privée et éventuellement à vérifier la signature d'Alice avec la clé publique d'Alice.

La stéganographie à clé publique est la technique de stéganographie qui garantit la sécurité maximale. Si un attaquant découvre la méthode de stéganographie utilisée, il ne pourra pas déchiffrer le message car il ne connait pas la clé privée du destinataire du message. De plus l'utilisation de la signature par clé publique ajoute l'authenticité du message. Cependant il peut tout de même se poser le problème d'une attaque de type *man in the middle*, si la cryptographie à clé publique n'est pas couplée avec un tiers de certification des clés publiques [2].

1.6 Caractéristiques de Schéma Stéganographique

Trois critères permettent de classer les algorithmes stéganographiques : La capacité, l'invisibilité et la robustesse [1].

- ***La capacité*** correspond à la taille de données pouvant être incorporées dans l'objet de couverture, relativement à la taille de celui-ci,
- ***L'invisibilité*** ou ***la transparence*** ou encore ***l'imperceptibilité*** qui dépend directement de la distorsion introduite par le processus de dissimulation pendant l'insertion de données ; la distorsion étant tout simplement le nombre de modifications ou de changements dans l'objet de couverture,
- ***La robustesse*** signifie la résistance de notre stégo-objet, c'est à dire rester normale même s'il subit des transformations (filtrage, etc....).

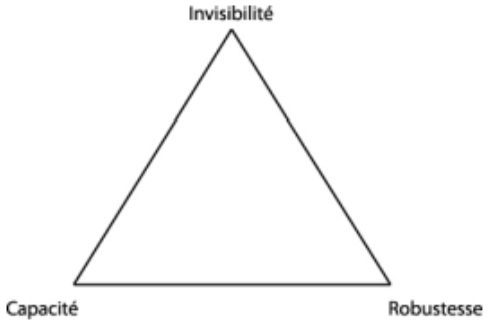


Figure 1.05 : *Compromis entre capacité, invisibilité et robustesse*

Ces trois critères ne peuvent pas être maximisés simultanément. Chacun d'entre eux aura une influence sur l'autre. Par exemple, la capacité va en contradiction avec la transparence.

1.7 La Stéganalyse

1.7.1 Définition

La steganalyse est l'ensemble des techniques qui permettent de détecter si un objet est un objet stégo ou non, c'est-à-dire s'il contient un message secret ou non, puis dans la mesure du possible de décoder ce message secret.

Pour contrer cette stéganalyse, un algorithme de stéganographie doit trouver un compromis entre l'invisibilité, c'est-à-dire la probabilité qu'un objet de couverture soit déclaré non stégo, la capacité, soit la quantité d'informations dissimulées dans un objet de couverture, et la robustesse, c'est-à-dire qu'un message secret reste intègre malgré des modifications de l'objet de couverture.

Dans le cadre de la stéganographie, ***l'invisibilité et la capacité sont les deux principales caractéristiques nécessaires***. La robustesse est quant à elle importante dans l'utilisation d'algorithmes de tatouage que nous décrirons plus tard.

1.7.2 Sécurité parfaite

La sécurité parfaite d'un système de stéganographie correspond à un système qui garantit l'invisibilité des messages secrets. Cette invisibilité dépend de la capacité. Ainsi plus la taille des messages secrets augmente par rapport à la taille des objets de couverture, plus grands sont les risques d'être détectés.

La définition formelle de la sécurité d'un système stéganographique est donnée par Cachin [4]. L'idée principale de sa définition est la sélection d'un objet de couverture en fonction d'une variable aléatoire C, avec une probabilité de distribution P_C. La dissimulation d'un message secret correspond à une fonction définie dans C.

Soit P_S la probabilité de distribution de E_K (c, m, k), qui est l'ensemble des stégo-objets produits par le système stéganographique. Si un objet de couverture c n'est jamais utilisé comme stégo-objet, alors $P_S = 0$. Voici la définition de l'entropie relative $D(P_1 \| P_2)$ entre deux distributions P_1 et P_2 définies sur l'ensemble $\mathbb{Q}$, qui détermine l'inefficience de la distribution P_2 sur la distribution P_1, où P_1 représente une probabilité de distribution aléatoire :

$$\mathbf{D(P_1 \| P_2) = \sum_{q \in Q} P_1(q) \log_2 \frac{P_1(q)}{P_2(q)}} \qquad (1.01)$$

P_C correspond alors à la distribution aléatoire P_1 et P_2 représente la probabilité P_S de distribution réelle du système de stéganographie. On peut ainsi représenter la sécurité d'un système de stéganographie en termes d'entropie relative $D(P_C \| P_S)$.

Sécurité parfaite pour un système de stéganographie : Soit G un système de stéganographie, P_S la probabilité de distribution de stégo-covers (médium de couverture) envoyés sur le canal de communication, et P_C la probabilité de distribution de C. G est dit ϵ***-securisé*** contre les attaquants passifs, si $D(P_C \| P_S) \leq \epsilon$, et parfaitement sécurisé si $\epsilon = 0$.

Puisque $D(P_C \| P_S)$ est nulle, cela signifie que l'entropie relative est égale à zéro, et donc que les deux probabilités de distribution sont égales, ce qui garantit que le système de stéganographie est théoriquement parfaitement sécurisé [2].

1.7.3 Détection des stégo-objets

Dans le cas de la détection des messages, Wendy doit décider si un objet de couverture c envoyé entre Alice et Bob contient un message secret ou non. Pour cela Wendy définit une fonction
$f : C \rightarrow 0, 1$

$$f(c) = \begin{cases} \mathbf{1} & \boldsymbol{c\ contient\ une\ secret} \\ \mathbf{0} & \boldsymbol{sinon} \end{cases} \quad (1.02)$$

Dans certains cas, Wendy va correctement détecter un stégo-objet ; dans d'autres cas, il ne va pas détecter un stégo-objet, on parle ***d'erreur de type II***. S'il détecte un objet non stégo comme un objet-stégo, on parle ***d'erreur de type I***.

En pratique, les systèmes de stéganographie cherchent à maximiser la probabilité β qu'un attaquant passif fasse des erreurs de type II, le système idéal étant pour β = 1 [2].

1.7.4 Techniques de stéganalyse

Les techniques de stéganalyse cherchent à minimiser la probabilité β qu'une attaque sur un système de stéganographie fasse des erreurs de type II. Ce qui correspond en fin de compte à maximiser la probabilité de détection de tous les stégo-objets.

Afin de mettre à jour l'utilisation de la stéganographie, les attaques possibles par un stéganalyste sur un système de stéganographie sont caractérisées en 6 types d'attaques :

- ***attaque sur objet stégo uniquement :*** seul l'objet stégo est connu par le stéganalyste.
- ***attaque sur objet de couverture connu :*** l'objet de couverture original (sans insertion de messages secret) est connu ainsi que l'objet stégo.
- ***attaque sur message connu :*** le stéganalyste connait le message
- ***attaque sur stéganographie choisie :*** l'algorithme de stéganographie ainsi que l'objet stégo sont connus.
- ***attaque sur messages secrets choisis :*** le stéganalyste est capable de générer des objets stégos avec des messages secrets choisis.
- ***attaque sur stéganographie connue :*** l'algorithme de stéganographie est connu, et le stéganalyste dispose de l'objet de couverture initial ainsi que l'objet stégo.

1.8 Tatouage numérique ou Watermarking

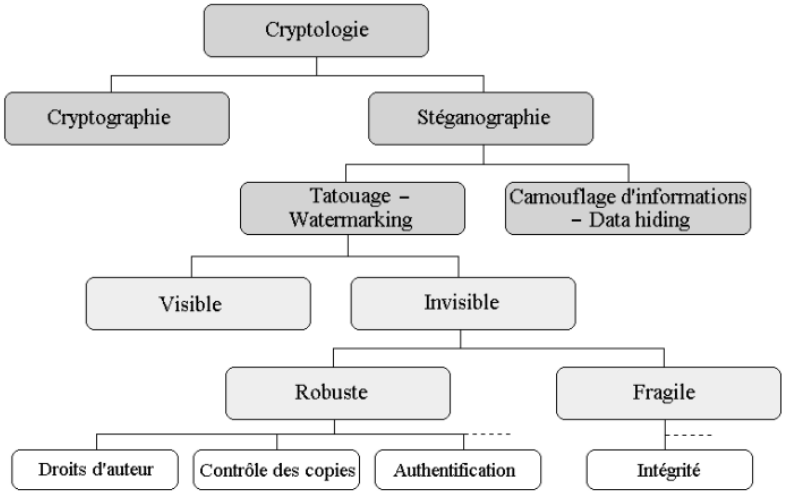


Figure 1.06 : *Diagramme de méthodes de sécurisation*

La ***cryptologie*** est la science qui permet de protéger des données. Elle regroupe les deux méthodes existantes de protection de l'information : la Cryptographie et la Stéganographie. Pour la stéganographie, deux types d'approches sont envisageables. La première consiste à cacher l'information à l'intérieur d'un autre document : ***Camouflage d'informations*** ou ***Data Hiding***. La seconde méthode d'utilisation de la stéganographie est d'intégrer une signature dans le document traité. Cette partie est appelée ***Tatouage*** ou ***Watermarking***.

1.8.1 Technologie du watermarking

Vers les années 1990, une nouvelle technologie est apparue à savoir le marquage numérique des signaux, appelée aussi tatouage numérique ou watermarking. La plus grande partie des applications de cette technologie est reliée à des applications de sécurité des données de médias audio et visuels, tel que la protection de droit d'auteur, le contrôle de l'intégrité des données, l'authentification, etc. Le processus de marquage comporte trois étapes [5] :

1. ***La génération de la marque :*** En général, la marque est un message transformé en une matrice W dont les valeurs sont binaires {±1} ou ternaires {1, 0, -1}.
2. ***L'insertion de la marque :*** Étant donné une image hôte I, l'insertion se fait soit dans le domaine spatial de l'image (dans les pixels), soit dans le domaine transformé de l'image (DCT ou Discrete Cosine Transform, DWT ou Discrete Wavelet Transform, …). De façon générale, l'insertion est une fonction qui prend en entrée l'image hôte I (ou image

originale) et la marque générée W, et délivre en sortie l'image marquée I'. Ce processus est illustré dans les figures 1.07 et 1.08.

3. ***La détection/extraction de la marque :*** Étant données une image test I' et une marque W, cette étape consiste à analyser l'image test I' et vérifier si la marque W est présente dans cette image. La vérification de la présence de la marque se fait par détection ou extraction :
    a. ***Détection :*** Mesure du degré de présence de la marque par une fonction de corrélation entre I' et W. La comparaison à un seuil de corrélation aboutit à une décision binaire (marque détectée/marque non détectée) (cf. figure 1.09).
    b. ***Extraction :*** Extraire les bits de la marque W' contenue dans l'image test I' et la comparer à la marque originale W en utilisant une métrique donnée tel que le TDR (True Detection Rate), signifiant le nombre de bits correctement extraits sur le nombre de bits total de la marque. Une autre métrique utilisée est le BER (Bit Error Rate) indiquant le ratio du nombre de bits extraits et erronés sur le nombre de bits total de la marque. Ce processus est illustré par la figure 1.10.

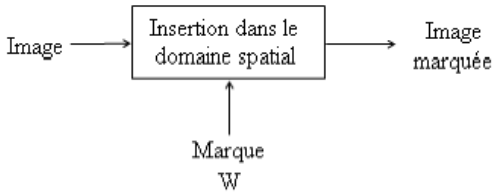


Figure 1.07 : *Représentation du processus d'insertion de la marque dans le domaine spatial*

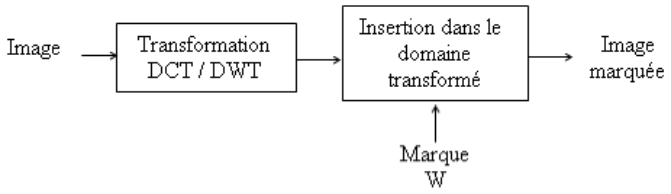


Figure 1.08 : *Processus d'insertion de la marque dans le domaine transformé DCT ou DWT*

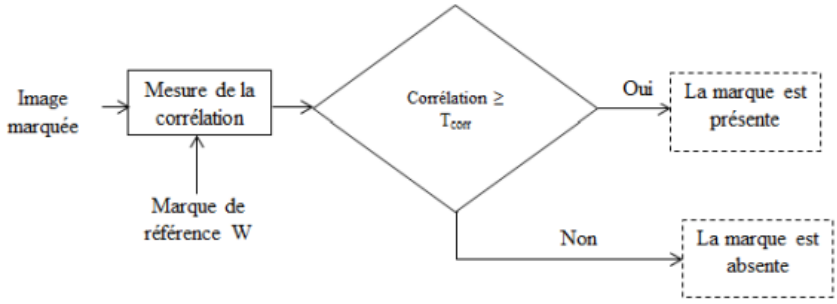


Figure 1.09 : *Représentation du processus de détection de la marque par corrélation*

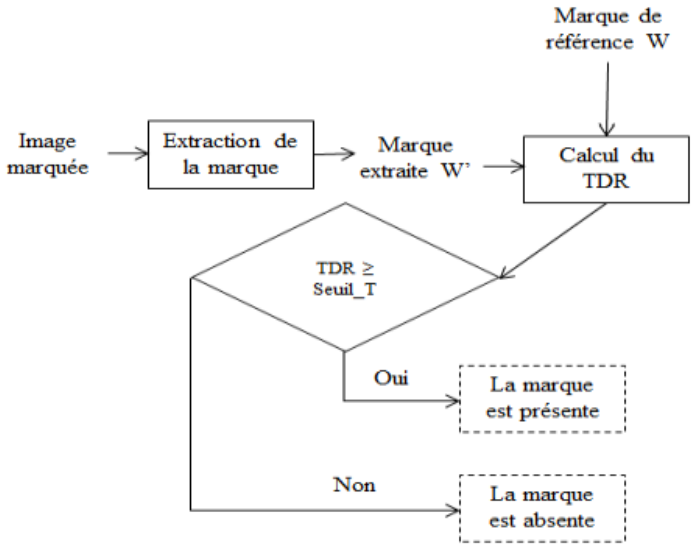


Figure 1.10 : *Processus d'extraction de la marque et comparaison avec la marque de référence*

1.8.2 Qualités d'un tatouage

Les performances d'un tatouage sont appréciées sous les critères principaux suivants :

- l'invisibilité,
- la robustesse,
- l'inversibilité,
- le ratio,
- la complexité,
- les informations nécessaires lors de la détection.

L'invisibilité : l'invisibilité d'une marque est sa capacité à être dissimulée sur un support. Cette invisibilité se traduit aussi par le respect de la qualité du document. Par exemple, dans le cas où le support serait une image, celle-ci ne doit pas être dégradée. Dans certains cas, une modification importante du contenu numérique peut avoir des conséquences limitées sur la perception visuelle que nous avons de l'image, et inversement.

Robustesse : Nous pourrions séparer cette rubrique en deux parties : ***la robustesse et la sécurité***. Ces deux caractéristiques sont souvent confondues surtout dans le cas du tatouage. Nous parlons de ***robustesse*** pour définir la ***résistance du tatouage face à des transformations de l'image tatouée***. Ces transformations peuvent être de type géométrique (rotation, zoom, découpage). Elles peuvent modifier certaines caractéristiques de l'image (histogramme des couleurs, saturation). Il peut aussi s'agir de tous les types de dégradations fréquentielles de l'image (compression avec pertes, filtres passe-haut ou passe-bas, passage analogique-numérique-analogique, etc.). Une

marque est robuste si elle est capable de résister aux attaques. En général, cette robustesse est plus ou moins importante suivant le choix du facteur d'insertion utilisé. Plus la force d'insertion est grande, plus la robustesse de la marque devrait être importante.

La sécurité caractérise la façon dont le marquage va résister à des attaques « malicieuses ». Nous pouvons faire des parallèles avec la cryptanalyse. Le pirate va chercher à laver l'image de façon intelligente. Il est sensé connaître l'algorithme et va, en général, chercher la clé qui lit le tatouage. Cela demande souvent une analyse approfondie de la technique de marquage employée.

L'inversibilité : L'inversibilité est la capacité d'un algorithme à extraire la marque de façon à restituer exactement l'image originale. Cette opération peut être utile par exemple en indexation. Les informations insérées dans le document peuvent être modifiées sans ajouter de dégradations au support ou de conflits dans les données insérées.

La complexité : La complexité indique le "nombre" et la nature des instructions algorithmiques nécessaires pour effectuer l'insertion de la marque ainsi que son extraction. Cette complexité va bien évidement indiquer le temps de calcul nécessaire à l'opération de tatouage. En particulier dans le cas de la vidéo, il sera préférable d'utiliser un algorithme peu complexe pour que les opérations d'un marquage ne soient pas coûteuses en temps de calcul.

La capacité ou ratio : Le ratio d'un système de tatouage numérique désigne le rapport : « nombre de données » à dissimuler sur « taille du document hôte ». Dans le cas du tatouage, généralement 16 à 64 bits sont suffisants pour assurer un service de droit d'auteurs. Pour des applications telles que l'indexation, l'algorithme de marquage devra être capable d'intégrer au message une marque contenant beaucoup plus d'informations. Dans certains cas, nous chercherons donc à intégrer une marque de grande capacité. De façon générale, plus le ratio est faible, plus la robustesse et l'imperceptibilité peuvent être élevées.

Informations nécessaires lors de la détection :

- ***Le tatouage privé***, aussi appelé ***non aveugle***, où la donnée originale est nécessaire à l'extraction. Ce système fonctionne en deux étapes : d'abord il compare le support marqué avec le support original ; ensuite un algorithme de décision est appliqué pour répondre si oui ou non la marque détectée correspond bien à la marque appliquée à l'origine. L'intérêt de ce marquage est très limité.
- ***Le tatouage semi-privé,*** aussi appelé ***semi-aveugle***, n'utilise pas la donnée originale, mais a besoin de la signature lors de l'extraction. Dans ce cas, il s'agit de répondre à la

question : « telle marque est-elle dans l'image ? ». La majorité des algorithmes de tatouage actuels utilise ce mode de fonctionnement.

- ***Le tatouage public***, aussi appelé ***aveugle***, ne nécessite ni la marque d'origine, ni le support d'origine. Ce type d'algorithme peut être difficile à mettre en place.

Afin de faire une synthèse, nous pouvons noter que l'invisibilité, la robustesse et la capacité sont fortement liées. En effet, d'une manière générale plus la marque est robuste et plus elle est visible. Ceci explique la grande difficulté à élaborer des techniques de tatouage, en particulier pour des applications liées à la sécurité.

1.8.3 Visibilité

Le principe fondamental du tatouage visible (cf. figure 1.11) consiste à dissimuler partiellement une image. Pour ce faire, il faudra utiliser un nombre indéterminé de marques visibles, qui ne pourront être efficacement effacées que si l'on possède une "clé secrète" adéquate. Ce type de tatouage est étudié actuellement pour gérer tout ce qui concerne les contrôles d'accès d'un unique document, correspondant en quelque sorte à une distribution de permission. On entend par contrôle d'accès la possibilité de restreindre la divulgation d'un document, en fonction de l'appartenance d'un utilisateur ou non la classe des "*ayants droit*" à la lecture de ce document.

Figure 1.11 : *Exemple de tatouage visible*

En pratique, l'intérêt d'un tatouage efficace réside dans son invisibilité (cf. figure 1.12). Et c'est d'ailleurs l'un des trois principaux critères d'un algorithme de marquage : faire en sorte que la différence avec l'original soit la plus minime possible [6].

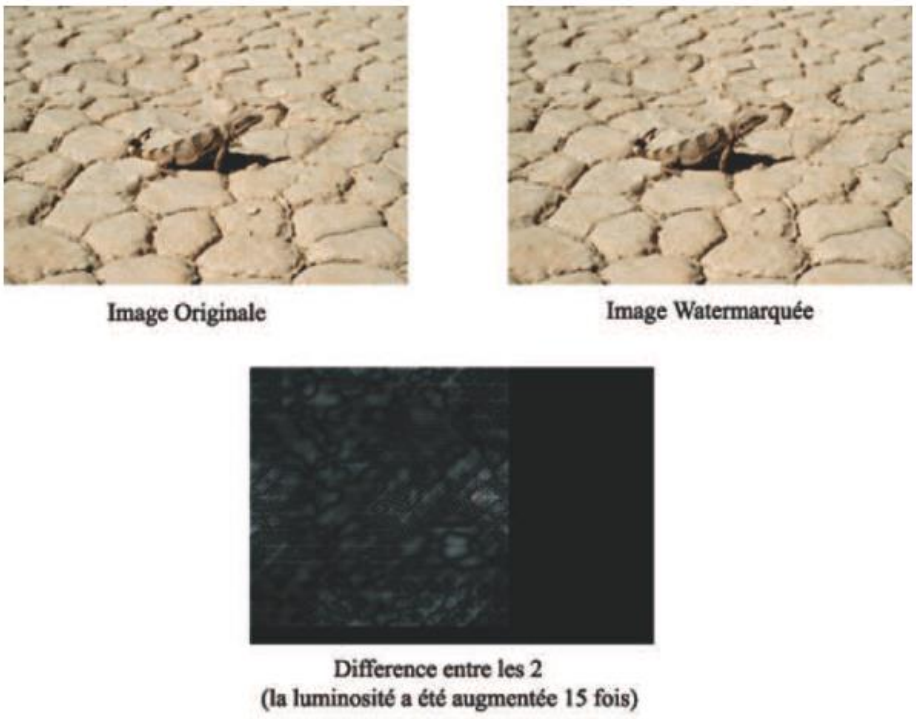


Figure 1.12 : *Exemple de tatouage (pseudo) invisible*

1.8.4 Robustesse et fragilité

Le deuxième principal critère de qualité d'un algorithme de tatouage concerne sa robustesse face à des manipulations de l'image : celui-ci doit pouvoir conserver l'information stockée dans le marquage en dépit de diverses transformations.

Or très peux d algorithme résiste à une simple compression ou un changement de format, mais ils sont sensés résister tout de même à des attaques basiques telle que des translations ou des rotations de l'image (souvent utilisées pour la mise en page).

Il est néanmoins intéressant de remarquer qu'il peut être utile, dans certain cas, de favoriser une fragilité plutôt qu'une robustesse (cf. figure 1.13).

Pour s'assurer par exemple de l'intégrité d'un document, le fait de le tatouer avec un algorithme fragile permettra, par la suite de vérifier si l'information marquée est toujours présente, ce qui sous-entend donc qu'elle n'a subi aucune modification malveillante (par exemple, une modification brutale de certaine partie textuelle). Cela permet donc une certaine falsification de

l'image. Ce critère de robustesse et de fragilité s'applique surtout à un marquage invisible (il n'y a aucun intérêt à se poser ce genre de question pour un tatouage visible).

Un cas intermédiaire, les Semi-fragiles : les tatouages semi-fragiles combinent à la fois les propriétés des marquages robustes et fragiles. Comme les robustes, ils tolèrent certains changements de l'image, comme des rotations, translations ou addition de bruit. Et comme les tatouages fragiles, ils sont capables de déterminer les régions où l'image a été brutalement modifiée et celles où elle reste authentique. Par conséquent les tatouages semi-fragiles arrivent à différencier les changements "légers" comme l'ajout d'un bruit à des changements « destructeurs ».

Ces algorithmes sont surtout très utiles par exemple dans le cas où une image marquée doit être diffusée sur le net, ou des types de compression comme le JPEG (Joint Photographic Experts Group), vont être employés. Un algorithme fragile ne supporterait pas ce type de transformation, et un robuste permettrait à quiconque récupèrerait cette image sur le net d'en modifier des parties sans que le marquage (donc le pseudo copyright servant à la falsification d'un document) soit altéré : tout le monde peut se présenter comme ayant un document original en sa possession. L'exemple d'un tatouage fragile (figure 1.13) permet d'illustrer aussi ce cas [6].

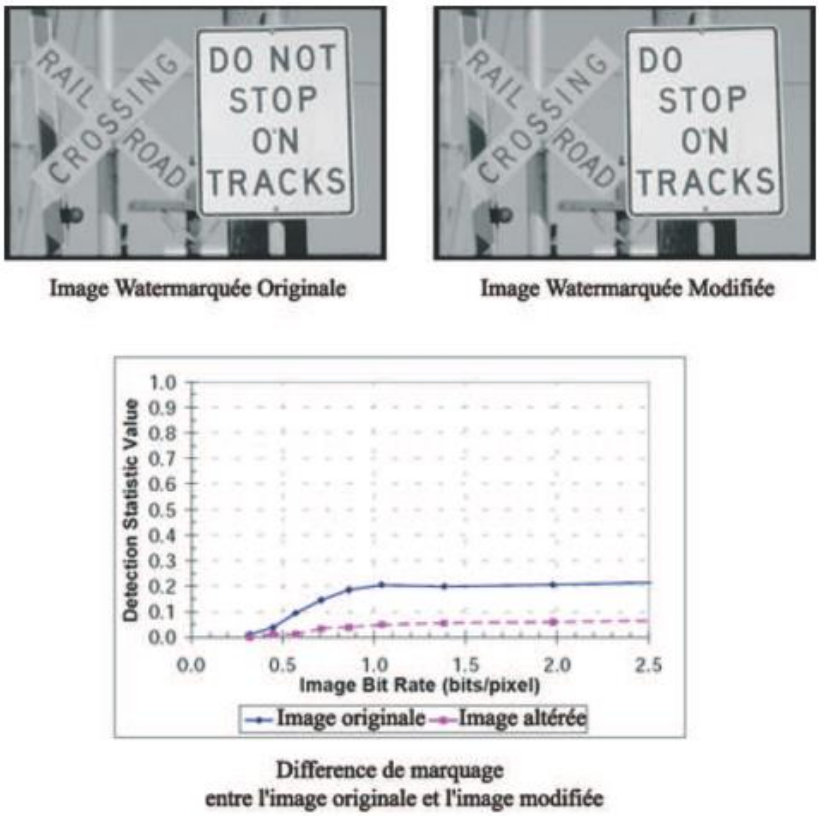


Figure 1.13 : *Exemple de tatouage fragile*

1.8.5 Applications

Plusieurs applications peuvent utiliser les techniques de tatouage. Pour chacune d'elles, les critères de qualité sont particuliers. Nous présentons ici les principales applications :

- droits d'auteurs,
- traçabilité (fingerprinting en anglais),
- protection contre les copies,
- authentification,
- indexation.

Droits d'auteurs : L'application la plus évidente du tatouage est le droit d'auteur. Le but est d'insérer une signature permettant d'identifier le propriétaire, de façon très robuste. Les deux principales qualités à respecter sont la robustesse et l'invisibilité de la marque.

Traçabilité (fingerprinting) : Le but de la traçabilité est de pouvoir contrôler et faire le suivi des copies de document. Cela implique de créer une marque originale pour chaque document distribué. Les qualités requises en termes de tatouage peuvent être alors classiquement la robustesse, l'invisibilité et le ratio afin d'éviter de fortes corrélations entre les diverses marques utilisées. Toutefois, nous pouvons envisager que dans certaines problématiques de traçabilité, le document transmis ne subit pas d'attaques malveillantes (restant dans un circuit privé). Dans ce cas, les qualités recherchées sont surtout l'invisibilité, mais aussi que l'insertion d'une nouvelle marque (par exemple pour chaque étape d'un processus), ne gomme pas les marques précédentes.

Protection contre les copies : La protection contre les copies consiste à intégrer au document une marque "intelligente", ou plus précisément un environnement matériel et/ou logiciel capable de communiquer avec la marque. Cela nécessite l'utilisation de matériels particuliers. En effet, les appareils doivent pouvoir détecter la marque et agir en conséquence, c'est-à-dire en permettant ou non la lecture, ou la copie du document. Par exemple dans le cadre de la copie de DVD (Digital Versatile Disc), le lecteur et enregistreur de DVD doit être en mesure de lire et modifier la marque du document. Dans ce cas, pour une marque particulière, l'appareil acceptera l'enregistrement, mais modifiera le contenu de la marque pour interdire toute copie ultérieurement. Cela permet de contrôler le nombre et les utilisateurs des copies.

Authentification : Dans le cadre d'application telle que l'authentification, le but est de détecter les modifications effectuées sur une donnée. Ce marquage est qualifié de "fragile". Il doit être résistant à des attaques classiques mais doit être détruit en cas de modification de la donnée. Dans

ce cas, une des solutions consiste à intégrer la marque sur les objets principaux de la donnée. Si un de ces objets est modifié ou supprimé, la marque est alors détruite.

Indexation

La marque intégrée au document dépend du contenu de celui-ci. L'algorithme utilisé devra permettre d'intégrer une marque de grande capacité. Par exemple, dans le cadre de la vidéo, la marque pourrait contenir la date d'enregistrement, le titre de la séquence, les noms des personnages ou objets principaux, etc. La marque devra être de grande capacité et invisible. La robustesse est à définir selon l'application mais n'est à priori qu'un problème secondaire.

1.9 Comparaison de la stéganographie, du tatouage et du fingerprinting

Voici une petite comparaison de la stéganographie (Data Hiding), du tatouage et de l'empreinte [7] :

Caractéristiques	**STÉGANOGRAPHIE** ***Steganography***	**TATOUAGE** ***Watermarking***	**EMPREINTE** ***Fingerprinting***
Données	Le message à transmettre	Une marque dépendant du support et/ou du propriétaire	Une empreinte dépendant du support et de son utilisateur
Support	Sans importance, le plus « banalisé » possible	Le document hôte dont on veut protéger les droits	Le document hôte dont on souhaite prévenir la diffusion de copie illégale
But de l'utilisateur	Cacher de l'information	Identifier l'émetteur (l'ayant droit)	Identifier le destinataire (l'utilisateur)
But de l'attaquant	Détecter les données et les extraire	Supprimer les données	Supprimer les données
Utilisation courante	Échapper à la censure	Copyrights, monitoring	Maîtrise de la diffusion

Tableau 1.01 : *Comparaison entre stéganographie, tatouage et empreinte*

1.10 Conclusion

Nous avons pu cerner à travers ce chapitre ce qu'est l'art de la stéganographie, et les points à savoir sur le sujet. En outre, nous avons pu distinguer la différence entre la stéganographie, le tatouage ou watermarking, et l'empreinte ou fingerprinting. Le regain d'intérêt actuel pour la stéganographie provient des restrictions imposées à la cryptographie. D'une manière générale, la stéganographie arrive en renforcement au chiffrement de données. Pour permettre de garantir une confidentialité maximum, les données sont tout d'abord chiffrées avant d'être dissimulées à l'aide d'un processus stéganographique. Le chapitre suivant concerne les techniques de stéganographie et de cryptage utilisées par notre application.

CHAPITRE 2
ALGORITHMES DE STEGANOGRAPHIE ET DE CRYPTAGE UTILISES

2.1 Introduction

Nous présentons en détails dans cette partie les techniques de stéganographie et de cryptage utilisées par notre application. Après une brève introduction des techniques de stéganographie modernes, nous insisterons surtout sur les techniques de stéganographie d'images. Ensuite, nous verrons aussi un algorithme de cryptage populaire, réputé difficilement cassable, qui sera utilisé lui aussi utilisé par l'application pour chiffrer les données avant d'appliquer la stéganographie.

2.2 Techniques stéganographiques modernes

Depuis les récents progrès en informatique, la stéganographie informatique est devenue à la mode, laissant de côté les autres techniques toujours aussi efficaces mais un peu tombées dans l'oubli, dans un monde qui vit à l'ère du numérique. La Stéganographie est une science pour cacher les données secrètes dans des fichiers informatiques comme couvertures, les fichiers les plus utilisés sont des fichiers textes, des images, des fichiers audio, etc....

Il est aussi possible de cacher des informations dans bien d'autres types de fichiers couramment échangés sur le réseau, telle que la vidéo ou bien dans des zones d'un disque dur inutilisées par le système de fichiers, et même dans des protocoles informatiques comme IP (Internet Protocol) et TCP (Transmission Control Protocol) [1].

Il existe plusieurs techniques pour mettre en place des schémas stéganographiques, cependant nous n'allons détailler que la méthode LSB (Least Significant Bit), et l'algorithme F5 pour les images, les techniques utilisées par notre application.

- La stéganographie sur images :
    - Usage des bits de poids faible d'une image (LSB)
    - Manipulation de la palette de couleurs d'une image
    - Message caché dans les choix de compression d'une image
- Dans un texte :
    - Modulation fine d'un texte
    - Marquage de caractères
    - Codage sous forme d'une apparence de spam
- dans un son,

- Fichiers HTML (Hypertext Markup Language),
- Canaux cachés http (Hypertext Transfer Protocol), canaux cachés IP, canaux cachés TCP, canaux cachés DNS (Domain Name System)…
- Rajout de données (EOF, en-têtes, ...) : les données cachées consistent en un fichier image rajouté juste sous le marqueur EOF (End Of File) d'un fichier.

2.2.1 LSB (Least Significant Bit)

La technique de base, dite LSB pour Least Significant Bit, est très simple. Dans le cas d'une image, elle consiste à modifier le bit de poids faible des pixels codant l'image. Une image numérique est une suite de points, que l'on appelle ***pixels***, et dont on code la couleur, le plus souvent, à l'aide d'un triplet d'octets (ex : RGB sur 24 bits, R = Red, G = Green, B = blue, les 3 couleurs primaires). Chaque octet du triplet ∈ [0, 255] peut être modifié de ± 1 sans que la teinte du pixel ne soit visuellement altérée. C'est ce que l'on fait en modifiant le bit de poids faible de l'octet [8].

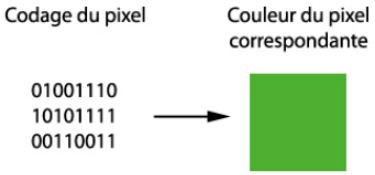


Figure 2.01 : *Codage RGB d'un pixel et couleur correspondante*

Imaginons les trois pixels suivants :

{R = 10110101, G = 11101010, B = 10010101},
{R = 11101010, G = 10110101, B = 00100100},
{R = 10110101, G = 11010101, B = 10101010}

On va cacher le caractère 'x' représenté par 88 dans le système ASCII (American Standard Code for Information Interchange). Le caractère 'x' correspond à la suite 01011000 comme représentation binaire. Alors, les trois pixels précédents seront modifiés par substitution du LSB pour insérer le message **01011000**.

{R = 1011010**0**, G = 1110101**1**, B = 1001010**0**},
{R = 1110101**1**, G = 1011010**1**, B = 0010010**0**},
{R = 1011010**0**, G = 1101010**0**, B = 1010101**0**}

2.2.2 L'algorithme F5

2.2.2.1 Introduction

L'algorithme F5 est probablement la première implémentation pratique d'une technique de matrix embedding en stéganographie (cf. annexe 1). Il a été inventé par A. Westfeld, et présenté en 1999 [10]. Le nom provient des précédents algorithmes développés par l'auteur : F3 et F4. F5 insère le message pendant une compression JPEG, et utilise les LSB des coefficients DCT pour réaliser l'opération de matrix embedding. L'algorithme est sûr face aux attaques visuelles et statistiques car il s'adapte à la distribution spécifique des coefficients DCT.

Dans un premier temps, nous décrirons brièvement l'algorithme de compression JPEG, puis nous verrons ensuite en détail le fonctionnement de l'algorithme F5 [9].

2.2.2.2 Compression JPEG

La norme JPEG est une norme qui définit le format d'enregistrement et l'algorithme de décodage pour une représentation numérique compressée d'une image fixe.

JPEG est l'acronyme de Joint Photographic Experts Group. Il s'agit d'un comité d'experts qui édite des normes de compression pour l'image fixe. La norme communément appelée JPEG, de son vrai nom ISO/IEC IS 10918-1 | ITU-T Recommandation T.81, est le résultat de l'évolution de travaux qui ont débuté dans les années 1978 à 1980 avec les premiers essais en laboratoire de compression d'images. Le groupe JPEG qui a réuni une trentaine d'experts internationaux, a spécifié la norme en 1991. La norme officielle et définitive a été adoptée en 1992 [1].

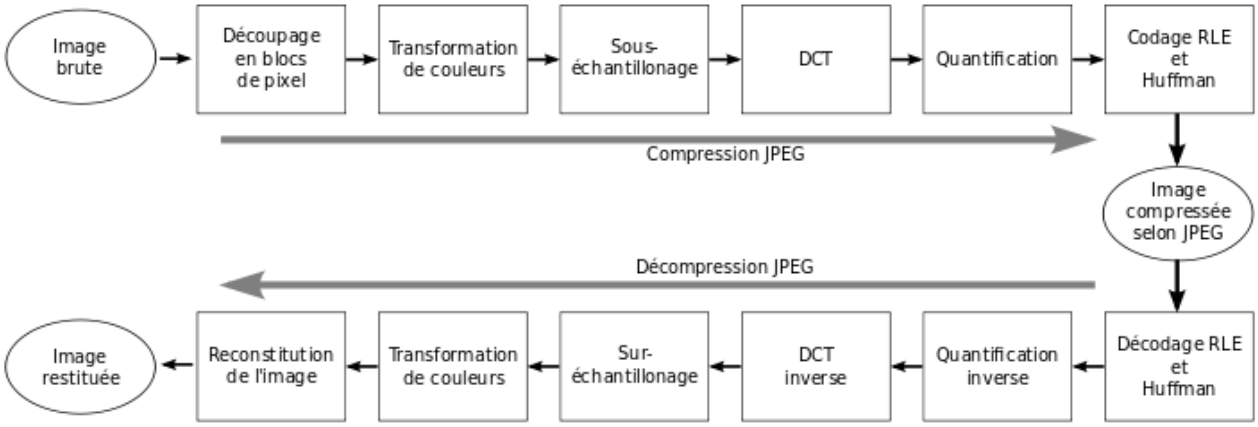


Figure 2.02 : *Organigramme de compression JPEG*

Transformation des couleurs : JPEG est capable de coder les couleurs sous n'importe quel format, toutefois ***les meilleurs taux de compression*** sont obtenus avec des codages de couleur de

type ***luminance/chrominance*** car l'œil humain est assez sensible à la luminance (la luminosité) mais peu à la chrominance (la teinte) d'une image. Afin de pouvoir exploiter cette propriété, l'algorithme convertit l'image d'origine depuis son modèle colorimétrique initial (en général RGB) vers le modèle de type chrominance/luminance YCbCr. Dans ce modèle, Y est l'information de luminance, et Cb et Cr sont deux informations de chrominance, respectivement le bleu moins Y et le rouge moins Y.

Soit une image I que l'on veut compresser au format JPEG. Chaque pixel p_i, est représenté par un triplet (R_i, G_i, B_i) dans l'espace RGB. La première étape de la compression JPEG consiste en un changement de l'espace des couleurs de RGB vers l'espace de couleurs YCbCr. Un pixel sera donc codé par un triplet (Y_i, Cb_i, Cr_i), où Y_i désigne la luminance, c'est-à-dire l'intensité lumineuse, Cb_i la chrominance bleue, c'est-à-dire l'intensité de la couleur bleue et Cr_i la chrominance rouge, c'est-à-dire l'intensité de la couleur rouge. Le changement d'espace s'effectue en appliquant les équations suivantes [1] :

Les équations de changement de RGB vers YCbCr :

$$\begin{pmatrix} \mathbf{Y} \\ \mathbf{Cb} \\ \mathbf{Cr} \end{pmatrix} = \begin{pmatrix} \mathbf{0.299} & \mathbf{0.587} & \mathbf{0.114} \\ \mathbf{-0.1687} & \mathbf{-0.3313} & \mathbf{0.5} \\ \mathbf{0.5} & \mathbf{-0.4187} & \mathbf{-0.0813} \end{pmatrix} \mathbf{x} \begin{pmatrix} \mathbf{R} \\ \mathbf{G} \\ \mathbf{B} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{128} \\ \mathbf{128} \end{pmatrix} \tag{2.01}$$

Les équations de changement de YCbCr vers RGB :

$$\mathbf{R = Y + 1.402(Cr - 128)} \tag{2.02}$$

$$\mathbf{G = Y - 0.34414(Cb - 128) - 0.71414(Cr - 128)} \tag{2.03}$$

$$\mathbf{B = Y + 1.772(Cb - 128)} \tag{2.04}$$

Le changement d'espace de couleurs se justifie par le fait que l'espace YCbCr est très proche du fonctionnement de l'œil humain. De plus, ce dernier est très sensible aux variations de luminance et très peu sensible aux variations de chrominance. De ce fait, on pourra effectuer des modifications sur les composantes Cb et Cr afin de compresser l'information visuelle et cela, sans que l'œil ne détecte la différence. Pour ce faire, les pixels sont regroupés en blocs de 4 × 4 pixels. Les quatre valeurs de chrominance bleue sont remplacées par leur moyenne ; la même transformation est effectuée sur les quatre valeurs de chrominance rouge. L'œil ne perçoit pas les changements effectués. Le gain de stockage est de 50% et la transformation appliquée est non

réversible, ***la compression est dite avec perte***. Cette transformation est appelée ***sous-échantillonnage*** [1].

Chacune des valeurs Y, Cb, Cr est un nombre codé sur P bits, compris entre 0 et 2^{P-1}, où P est appelé ***précision***. Les valeurs Y, Cb et Cr sont alors ramenées sur l'intervalle $[-2^{P+1}+1, 2^{P-1}-1]$ par une translation de -2^{P-1}.

Transformation DCT : la transformée DCT (Discrete Cosine Transform, en français transformée en cosinus discrète), est une transformation numérique qui est appliquée à chaque bloc. Cette transformée est une variante de la transformée de Fourier. Elle décompose un bloc, considéré comme une fonction numérique à deux variables, en une somme de fonctions cosinus oscillant à des fréquences différentes. Chaque bloc est ainsi décrit en une carte de fréquences et en amplitudes plutôt qu'en pixels et coefficients de couleur. La valeur d'une fréquence reflète l'importance et la rapidité d'un changement, tandis que la valeur d'une amplitude correspond à l'écart associé à chaque changement de couleur. Un coefficient DCT S_{uv} de coordonnées (u, v) dans le domaine fréquentiel s'exprime en fonction des 64 valeurs du bloc dans le domaine spatial s_{xy} de coordonnées (x, y) à partir de la formule [1] :

$$\mathbf{S_{uv} = \frac{1}{4} C(u)C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} S_{xy} \cos\left[\frac{(2x+1)u\pi}{16}\right] \cos\left[\frac{(2y+1)v\pi}{16}\right]} \qquad (2.05)$$

Avec $\mathbf{C(0) = \frac{1}{\sqrt{2}}}$ et $\mathbf{C(u) = 1}$ si $\mathbf{u \neq 0}$

Cette transformation est en fait la partie réelle de la transformée de Fourrier discrète. La transformée en cosinus discrète inverse, (IDCT ou Inverse Discrete Cosine Transform) permet de retrouver, lors de la décompression, les blocs dans le domaine spatial à partir des coefficients DCT, à l'aide de la formule :

$$\mathbf{S_{xy} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v)S_{uv} \cos\left[\frac{(2x+1)u\pi}{16}\right] \cos\left[\frac{(2y+1)v\pi}{16}\right]} \qquad (2.06)$$

Parmi les coefficients DCT, on distingue le coefficient $S_{0,0}$, aussi appelé coefficient DC, les autres que l'on nomme coefficients AC. Le coefficient DC est le coefficient des basses fréquences, c'est donc lui qui est porteur de la majorité de l'information. Ce coefficient étant généralement le plus grand ; on ne stocke que sa différence avec le coefficient DC du bloc précédent. Le tableau (2.01) illustre un bloc de coefficients DCT.

235,6	-1	-12,1	-5,2	2,1	-1,7	-2,7	1,3
-22,6	-17,5	-6,2	-3,2	-2,9	-0,1	0,4	-1,2
-10,9	-9,3	-1,6	1,5	0,2	-0,9	-0,6	-0,1
-7,1	-1,9	0,2	1,5	0,9	-0,1	0,6	-0,1
0,6	-0,8	1,5	1,6	-0,1	-0,7	0,6	1,3
1,8	-0,2	1,6	-0,3	-0,8	1,5	1	1
-1,3	-0,4	-0,3	-1,5	-0,5	1,7	1,1	-0,8
-2,6	1,6	-3,8	-1,8	1,9	1,2	-0,6	-0,4

Tableau 2.01 : *Exemple d'un bloc de coefficients DCT*

Le coin en haut à gauche des blocs DCT contient les valeurs de basses fréquences, tandis que les hautes fréquences se situent dans le coin bas à droite. Dans un bloc de coefficients DCT, les droites d'équation u + v = cte regroupent les coefficients pour une fréquence donnée [1].

Quantification : La quantification est l'étape de l'algorithme de compression JPEG au cours de laquelle se produit la majeure partie de la perte d'information (et donc de la qualité visuelle), mais c'est aussi celle qui permet de gagner le plus de place (contrairement à la DCT, qui ne compresse pas). La DCT a retourné, pour chaque bloc, une matrice de 8 × 8 nombres (dans l'hypothèse que les blocs de l'image font 8 × 8 pixels). La quantification consiste à diviser cette matrice par une autre, appelée matrice de quantification, et qui contient 8 × 8 coefficients spécifiquement choisis par le codeur. Le but est ici d'atténuer les hautes fréquences, c'est-à-dire celles auxquelles l'œil humain est très peu sensible. Ces fréquences ont des amplitudes faibles, et elles sont encore plus atténuées par la quantification ; certains coefficients sont même souvent ramenés à 0. Le calcul permettant la quantification est le suivant : Les tables de quantification *tab − quant* sont calculées à partir des tables de référence *tab − ref* (une pour la luminance et une pour les chrominances) à partir de la formule :

$$\mathbf{tab-quant(i,j)} = \left\lceil \frac{\left[\mathbf{tab-ref(i,j)\times c(Q)+50}\right]}{\mathbf{100}} \right\rceil, \forall \mathbf{i,j=0,\ldots,7} \qquad (2.07)$$

Où c(Q) se déduit à partir du facteur de qualité de la manière suivante :

$$\boldsymbol{c(Q)} = \begin{cases} \frac{\mathbf{5000}}{\boldsymbol{Q}} & \boldsymbol{si\ Q < 50} \\ \mathbf{200 - 2Q} & \boldsymbol{sinon} \end{cases} \qquad (2.08)$$

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Tableau 2.02 : *Tables de référence pour la luminance (à gauche) et la chrominance (à droite)*

Le facteur de qualité prend des valeurs comprises entre 1 et 100, la valeur 1 correspondant à une faible qualité, c'est-à-dire de dégradation maximale et la valeur 100 à la qualité la plus forte, c'est-à-dire sans dégradation. En effet, si Q = 100, alors les coefficients des tables de quantification sont tous égaux à 1 d'après la valeur c(Q) ; les coefficients DCT sont juste arrondis lors de l'étape de quantification. La quantification a pour effet de favoriser les basses fréquences, c'est-à-dire celles qui contiennent le plus d'information [1].

En effet, les coefficients les plus faibles se trouvent en haut à gauche des tables et les coefficients les plus grands en bas à droite. De la même manière, les coefficients DCT de la luminance sont favorisés par rapport à ceux des composantes de chrominance. En conclusion, la quantification introduit majoritairement des coefficients DCT quantifiés à 0 dans les hautes fréquences et les composantes de chrominance.

Codage RLE : L'étape de quantification introduit beaucoup de zéros dans les hautes fréquences. Un codage particulier, le Run Length Encoding (RLE) permet de compresser sans perte, des données qui comportent majoritairement des longues plages de symboles identiques. Pour préparer au codage RLE et regrouper au maximum les 0, le format JPEG prévoit le regroupement des coefficients AC quantifiés suivant une séquence dite Zigzag. Cette séquence de parcours des blocs DCT quantifiés est représentée sur la figure 2.03. Elle consiste à regrouper tout d'abord les coefficients AC quantifiés d'une même fréquence puis de les ordonner selon les fréquences croissantes [1].

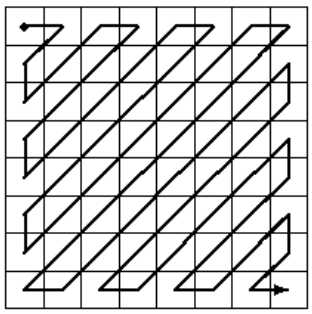


Figure 2.03 : *Séquence Zig-Zag*

Le codage RLE, représente chaque coefficient AC quantifié par un triplet (RL, S, V), où :

- RL de 0 à 16, codé sur 4 bits, compte le nombre de 0 précédant le coefficient à coder,
- S de 1 à 10, codé sur 4 bits, est le nombre de bits nécessaires pour coder le coefficient,
- V est la valeur du coefficient codée sur S bits.

L'étude des distributions des triplets (RL, S, V) montre que les valeurs V semblent distribuées aléatoirement, tandis que certaines paires RL et S apparaissent plus fréquemment que d'autres. Le format JPEG prévoit alors un codage entropique pour compresser sans perte les paires RL et S. Suivant les modes du format JPEG, ce codage entropique peut être un codage de Huffman ou un codage arithmétique. Le codage de Huffman étant majoritairement utilisé, nous ne détaillerons que celui-ci. Le lecteur intéressé par le codage arithmétique pourra se référer à [11]. Les coefficients DC sont les termes de plus basse fréquence ; généralement non nuls, ils sont codés différemment des coefficients AC. De grande amplitude, seule la différence entre deux coefficients DC successifs est codée. Cette différence est ensuite codée par une paire (S, V) avec S de 0 à 11, codé sur 4 bits représentant le nombre de bits pour coder V et V la valeur de la différence codée sur S bits. Seule la valeur S sera ensuite codée par l'algorithme de Huffman [1].

Codage de Huffman : le codage de Huffman [12] est un codage à longueur variable, c'est-à-dire dont les mots du code sont de longueur variable. De tels codes sont aussi appelés codage entropique. De plus, celui-ci est dit préfixé, c'est-à-dire que chacun des mots du code ne peut être le début d'un autre mot du même code. Nous allons décrire l'algorithme de construction du code de Huffman illustré par un exemple adapté de [12]. Pour plus de détails, le lecteur pourra se référer à [12].

Le code de Huffman pour la séquence S = « abracadabra » est représenté dans le tableau 2.03. Le décodage de l'algorithme se fait en lisant les lettres du code jusqu'à trouver un mot de code

présent dans le dictionnaire. Le code étant préfixé, aucune ambiguïté sur le mot de code n'est possible et le décodage est univoque. Le dictionnaire étant construit en fonction des données à coder, il doit être fourni au décodeur [1].

Algorithme de Huffman

Entrée : Une séquence S de N symboles.

Sortie : Code de Huffman.

Début de l'algorithme :

Ordonner les N symboles par ordre croissant de leur nombre d'occurrences dans S, chaque couple (symbole, occurrence) étiquetant un arbre réduit à sa racine.

Tant qu'il reste plus d'un arbre binaire

Retirer de la liste les deux arbres binaires de poids minimum

Fusionner ces arbres en un troisième de poids la somme des poids de ces fils

Affecter à l'arc vers le fils de droite la valeur 1 et vers celui de gauche la valeur 0

Insérer ce nouvel arbre dans la liste

Fin tant que

Affecter à chaque feuille la concaténation de la valeur des arcs depuis la racine

return pour chaque feuille, le couple (symbole, valeur binaire).

symbole	code
a	1
b	01
r	000
c	0010
d	0011

Tableau 2.03 : *Code de Huffman pour S = "abracadabra"*

La dernière étape du format JPEG est un codage entropique de Huffman de la valeur S de la différence des coefficients DC et des paires (RL, S) pour les coefficients AC. Deux cas sont alors possibles. Soit les dictionnaires sont transmis dans l'entête du fichier JPEG pour le décodage, soit ce sont les numéros de dictionnaires prédéfinis par la norme. Ces dictionnaires prédéfinis ont été déterminés expérimentalement par le groupe d'experts JPEG et sont en moyenne très performants. La plupart des applications manipulant le format JPEG utilisent les dictionnaires prédéfinis [1].

2.2.2.3 Description de l'algorithme F5

L'algorithme F5 d'A. Westfeld est un algorithme ±1 sur les coefficients DCT quantifiés non nuls d'une image (pendant une compression JPEG), en utilisant la technique de matrix embedding (cf. annexe 1). Le seul artéfact produit est une augmentation du nombre de coefficients à 0, une caractéristique ressemblant simplement à une compression JPEG de qualité inférieur [9].

L'algorithme présenté par A. Westfeld prend 5 paramètres en entrée pour insérer un message :

- une image servant de support (compressée ou non).
- un message à être inséré.
- le facteur de qualité à être utilisé lors de la quantification.
- un mot de passe afin de générer une marche aléatoire sur le support et le message.
- un commentaire pouvant être inséré dans l'en-tête du fichier JPEG.

Il prend deux paramètres en entrée afin d'extraire un message :

- un stégo-image.
- le mot de passe utilisé lors de l'insertion.

Voici le fonctionnement de l'algorithme d'insertion :

1. Effectuer une compression JPEG de l'image, arrêter juste après la quantification.
2. Calculer la capacité C de l'image et en déduire le paramètre p pour effectuer la technique de matrix embedding (cf. annexe 1).
3. Générer une marche pseudo-aléatoire afin de mélanger le support et le message.
4. Insérer dans les premiers bits du support la taille du paramètre p utilisé ainsi que la taille du message, en utilisant simplement une technique LSB.
5. Insérer des morceaux de message de longueur p dans des morceaux de support (coefficients AC non nul) de longueur $2^p - 1$ en utilisant la technique de matrix embedding. Afin de changer la valeur d'un bit, la valeur absolue du coefficient est décrémentée de 1. Si le coefficient devient 0, cette situation est appelée effondrement, et les p bits devant être insérés sont insérés à nouveau en retournant à l'étape 5.
6. Si tout le message a pu être inséré, alors « processus réussi », sinon « warning » affiché.

Lors de l'extraction, le récepteur n'a alors plus qu'à générer la marche aléatoire, retrouver les paramètres de l'algorithme utilisés, et de calculer le syndrome de chaque bloc de coefficients AC non nuls afin de reconstruire le message. La capacité C à l'étape 2 est obtenu par la formule 2.09.

$$C = h_{DCT} - \frac{h_{DCT}}{64} - h(0) - h(1) + 0.49h(1) \quad (2.09)$$

Avec h_{DCT} le nombre de coefficients DCT, $\frac{h_{DCT}}{64}$ le nombre de coefficients DC, $h(0)$ le nombre de coefficients à 0, $h(1) - 0.49h(1)$ le nombre de coefficients perdus par effondrement.

2.3 Cryptographie

La cryptographie est la science qui utilise les mathématiques pour chiffrer et déchiffrer des données. La cryptographie permet de stocker des informations sensibles ou de les transmettre à travers des réseaux non sûrs (comme Internet) de telle sorte qu'elles ne puissent être lues par personne à l'exception du destinataire convenu.

Les données qui peuvent être lues et comprises sans mesures spéciales sont appelées *texte clair (ou libellé)*. Le procédé qui consiste à dissimuler du texte clair de façon à cacher sa substance est appelée *chiffrement* (dans le langage courant on parle plutôt de *cryptage* et de ses dérivés : *crypter*, *décrypter*). Chiffrer du texte clair produit un charabia illisible appelé *texte chiffré (ou cryptogramme)*. Le processus de retour du texte chiffré à son texte clair originel est appelé *déchiffrement* [13].

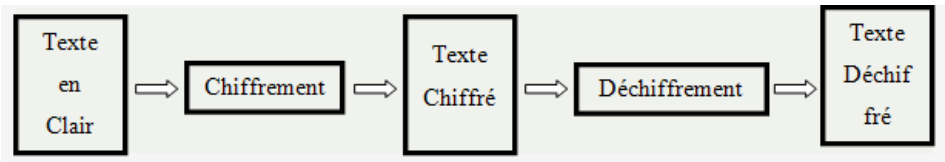


Figure 2.04 : *Chiffrement et déchiffrement*

2.3.1 Chiffrement à clé privée

Le chiffrement à clé privé consiste à utiliser la même clé pour le chiffrement et le déchiffrement. Par analogie c'est le principe d'une serrure d'une porte : tous les utilisateurs autorisés ont une clé identique. On distingue le système de chiffrement en continu et chiffrement par bloc [13].

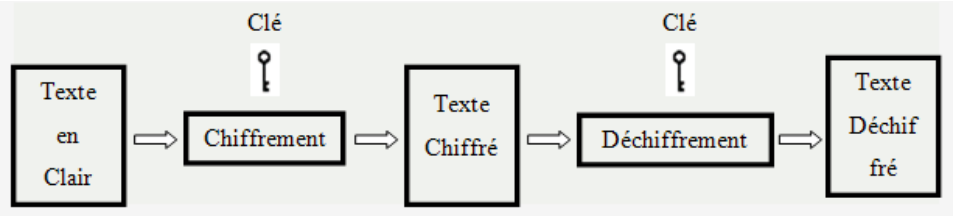


Figure 2.05 : *Chiffrement à clé privée*

2.3.2 Chiffrement à clé publique

Les problèmes de distribution des clés sont résolus par la cryptographie à clé publique ou cryptographie asymétrique. Ce concept a été introduit par Whitfïeld Diffie et Martin Hellman en 1975. La cryptographie à clé publique est un procédé asymétrique utilisant une paire de clés asymétrique associé : ***une clé publique qui crypte des données*** et ***une clé privée ou secrète correspondante pour le décryptage***. La clé publique peut être ainsi publiée tout en conservant sa clé privée secrète. Il est basé sur une méthode mathématique garantissant un encryptage facile et rapide, et un décryptage difficile. S'il fallait aussi une analogie, considérons que l'on crypte le message avec un cadenas (clé publique) que le détenteur de la clé privée peut ouvrir pour lire le cryptogramme. Il est impossible de retrouver la clé privée à partir de la clé publique [13].

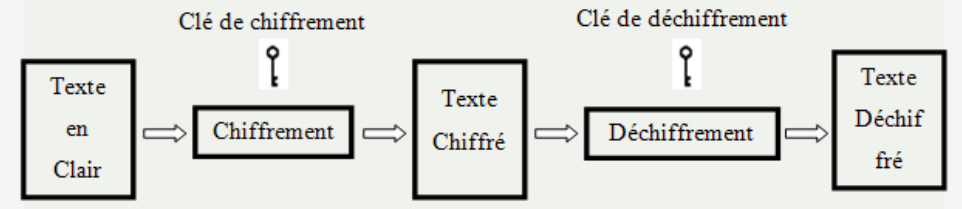


Figure 2.06 : *Chiffrement à clé publique*

2.3.3 L'advanced Encryption Standard ou « AES »

L'AES aussi connu sous le nom de Rijndael est le standard de chiffrement symétrique destiné à remplacer le « DES » ou Data Encryption Standard (FIPS P 46-3) choisi comme standard dans les années 1970. Il remporta en octobre 2000 le concours AES, lancé en 1997 par le NIST (National Institute of Standards and Technology) et devint le nouveau standard de chiffrement pour les organisations du gouvernement des États-Unis. Il est aujourd'hui défini dans le « FIPS P 197 » (« Federal Information Processing Standards Publication ») [14].

Pour AES les blocs de données en entrée et en sortie sont des blocs de **128 bits**, c'est à dire de **16 octets**. ***Les clés secrètes*** ont, suivant la version du système : ***128*** bits (16 octets), ***192*** bits (24 octets) ou ***256*** bits (32 octets).

Tailles des clés AES	**Nombre des clés AES**	**Rapport de 10^{21} avec le DES**
128	$3{,}4 \times 10^{38}$	1 clé DES cassée en 1 seconde correspond à 149 milles milliards d'années pour casser une clé AES sur 128 bits (Age de l'univers : 20×10^{9} années)
192	$6{,}2 \times 10^{57}$	
256	$1{,}1 \times 10^{77}$	
Taille de la clé DES	**Nombre des clés DES**	
56	$7{,}2 \times 10^{16}$	

Tableau 2.04 : *Comparaison de la taille des clés pour AES et DES [14]*

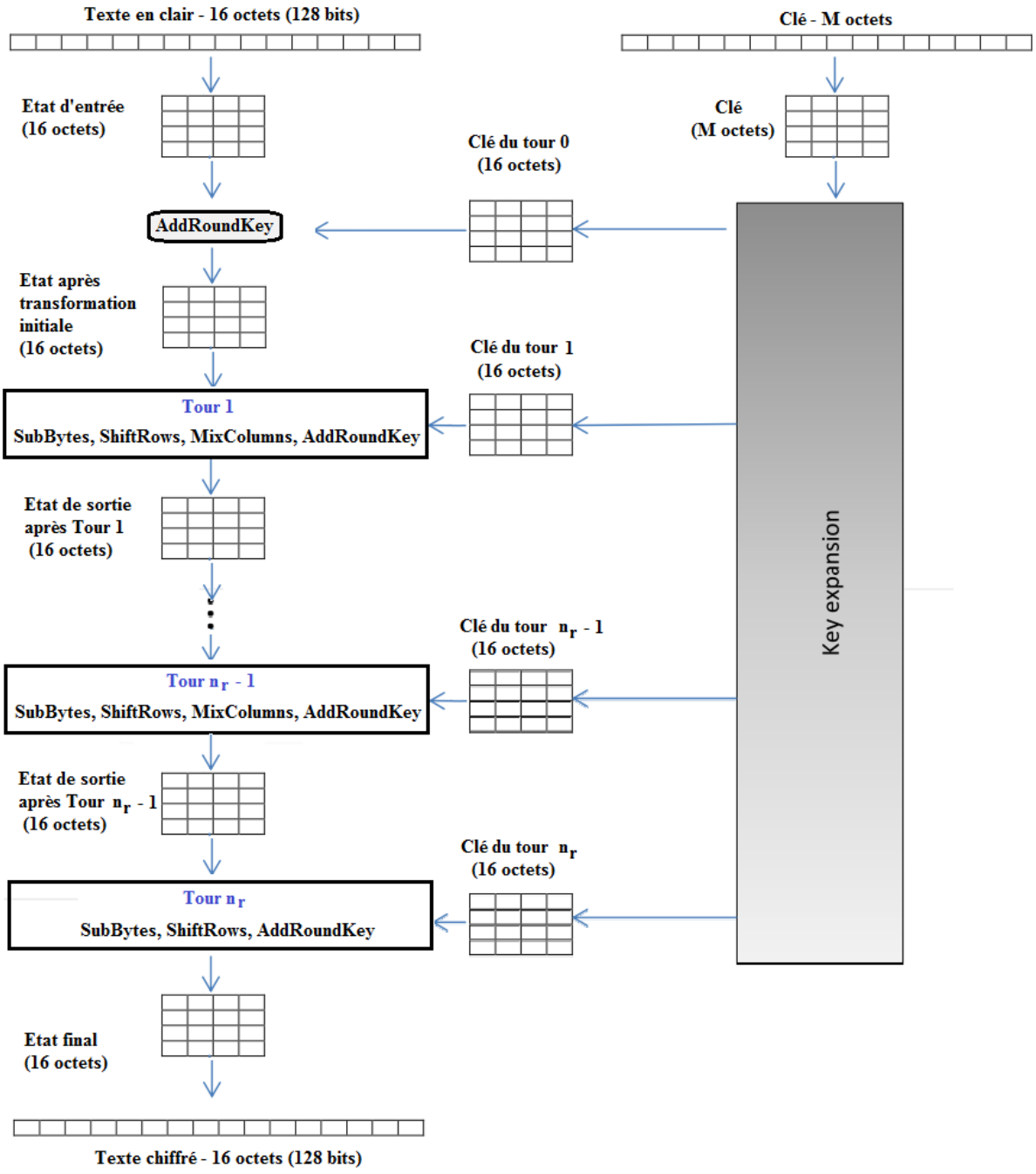


Figure 2.07 : *Fonctionnement global du chiffrement AES par bloc de 128 bits*

On découpe les données et les clés en octets et on les place dans des tableaux. Les données comportent t_d = 16 octets p_0, p_1, …, p_{15} qui sont classés dans un tableau ayant 4 lignes et 4 colonnes. Le tableau est rempli colonnes par colonnes.

De même la clé est découpée en octets (t_k = 16, t_k = 24 ou t_k = 32 octets) $\mathbf{k_0, k_1, ..., k_{t_k-1}}$. Ces octets sont aussi classés dans un tableau de 4 lignes et N_k colonnes (N_k = 4, N_k = 6 ou N_k = 8).

p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

k_0	k_4	k_8	k_{12}	k_{16}	k_{20}	k_{24}	k_{28}
k_1	k_5	k_9	k_{13}	k_{17}	k_{21}	k_{25}	k_{29}
k_2	k_6	k_{10}	k_{14}	k_{18}	k_{22}	k_{26}	k_{30}
k_3	k_7	k_{11}	k_{15}	k_{19}	k_{23}	k_{27}	k_{31}

Tableau 2.05 : *Données et clés (cas N_k = 8)*

Le système AES effectue ***plusieurs tours*** d'une même composition de transformations.

2.3.3.1 Le nombre de tours

Suivant la version (la taille de la clé), ce nombre de tours noté n_r est différent. Le nombre n_r est donné dans le tableau suivant.

N_k	4	6	8
n_r	10	12	14

Tableau 2.06 : *Valeurs du nombre de tours n_r suivant N_k*

2.3.3.2 La clé de tour

À partir de la clé initiale K, le système crée n_r + 1 clés de tour ayant chacune 16 octets. Ces clés seront stockées dans un tableau unidimensionnel TK et seront notées TK[0], TK[1], · · · TK[n_r]. Nous verrons ultérieurement comment sont calculées ces clés en fonction de la clé K du système.

2.3.3.3 Vue globale du fonctionnement

La procédure suivante décrit le fonctionnement global du système AES. Elle prend en entrée un tableau de données St (texte clair) qui est modifié par la procédure et renvoyé en sortie (texte chiffré).

Entrée : le tableau St et la clé K

Sortie : le tableau St modifié

AES(St, K)

début

KeyExpansion(K, TK) ;

AddRoundKey(St, TK[0]) ;

pour (i = 1 ; i < n_r ; i++) Round(St, TK[i]) ;

FinalRound(St, TK[n_r]) ;

fin

Les procédures appelées Round et FinalRound sont elles-mêmes composées comme suit :

Entrée : le tableau d'état St et une clé de tour T

Sortie : le tableau St modifié

```
Round(St, T )
début
SubBytes(St) ;
ShiftRows(St) ;
MixColumns(St) ;
AddRoundKey(St, T ) ;
fin
```

Entrée : le tableau d'état St et une clé de tour T

Sortie : le tableau St modifié

```
FinalRound(St, T)
début
SubBytes(St) ;
ShiftRows(St) ;
AddRoundKey(St, T ) ;
fin
```

2.3.3.4 La procédure SubBytes

Cette procédure est la seule transformation qui ne soit pas linéaire. C'est donc grâce à celle-ci que le système est résistant. Elle utilise une opération sur le corps fini à 256 éléments.

Le corps fini à 256 éléments : Considérons le polynôme $P(X) = X^8 + X^4 + X^3 + X + 1$. Ce polynôme à coefficients dans le corps à 2 éléments $F_2 = \{0,1\}$ est irréductible sur ce corps. Les éléments du corps à 256 éléments seront les octets $b_7b_6b_5b_4b_3b_2b_1b_0$ considérés comme des polynômes $b(X) = b_7X^7 + b_6X^6 + b_5X^5 + b_4X^4 + b_3X^3 + b_2X^2 + b_1X + b_0$, ce qui nous permet de définir les deux opérations suivantes :

Addition : $a_7a_6a_5a_4a_3a_2a_1a_0 + b_7b_6b_5b_4b_3b_2b_1b_0 = c_7c_6c_5c_4c_3c_2c_1c_0$, avec $c(X) = a(X) + b(X)$, ce qui donne aussi $\mathbf{c_i = a_i \oplus b_i}$.

Multiplication : $a_7a_6a_5a_4a_3a_2a_1a_0 \times b_7b_6b_5b_4b_3b_2b_1b_0 = c_7c_6c_5c_4c_3c_2c_1c_0$ avec $c(X) = a(X) \times b(X) \bmod P(X)$.

On a ainsi une structure de corps et donc tout élément non nul est inversible. Nous noterons g l'application de F_{256} dans F_{256} définie par :

$$g(x) = \begin{cases} 0 & si\ x = 0 \\ x^{-1} & sinon \end{cases} \tag{2.10}$$

L'inverse d'un élément b(X) se trouve par l'algorithme d'Euclide étendu.

La fonction affine f : Définissons b = *f*(a) grâce à une matrice

$$\begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{2.11}$$

La matrice carrée qui intervient est une matrice circulante, elle correspond donc à une multiplication de polynômes modulo $\mathbf{X^8 - 1}$.

$b(x) = ((X^4 + X^3 + X^2 + X + 1) \times a(X) \bmod (X^8 + 1)) + (X^6 + X^5 + X + 1)$.

On remarque que $g^{-1} = g$ et que f^{-1} est dénie par :

$$\begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \tag{2.12}$$

La procédure SubBytes : On définit alors

$$\mathbf{s(a) = \mathit{f}(g(a))} \tag{2.13}$$

On a donc aussi

$$\mathbf{s^{-1}(b) = g(}\boldsymbol{f}^{\mathbf{-1}}\mathbf{(b))} \qquad (2.14)$$

La procédure ***SubBytes*** applique **s** à chaque octet de l'entrée **St**.

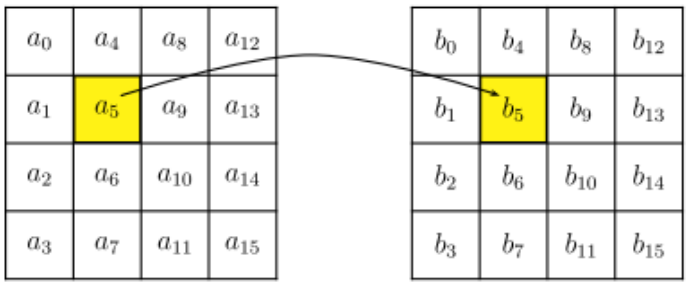


Figure 2.08 : *Transformation SubBytes*

La procédure SubBytes peut en outre être simplifiée en utilisant une S-Box (cf. annexe 3).

2.3.3.5 La procédure ShiftRows

La procédure consiste à opérer une rotation à gauche sur chaque ligne du tableau d'entrée. Le nombre de cases dont on décale la ligne i ($0 \leq i \leq 3$) est de i.

La transformation inverse est immédiate à calculer.

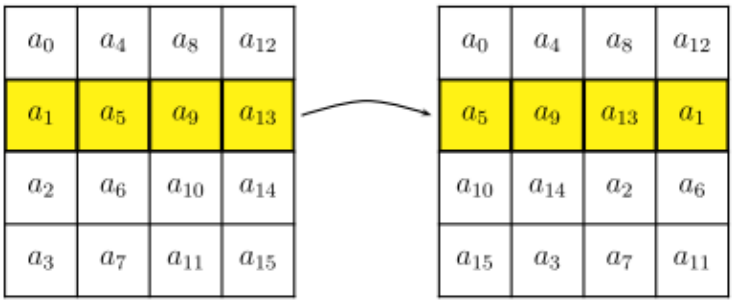


Figure 2.09 : *Transformation ShiftRows*

2.3.3.6 La procédure MixColumns

La transformation MixColums consiste à appliquer à chaque colonne du tableau des données une même transformation que nous allons décrire.

Considérons une colonne $c = (c_1, c_2, c_3, c_4)^t$. Les éléments c_i sont des éléments de F_{256}. Chaque colonne **c** est transformée en une colonne **d** grâce à une transformation linéaire (cf. figure 2.10) donnée par sa matrice dont les coefficients sont dans F_{256} et que nous écrivons comme des octets en hexadécimal.

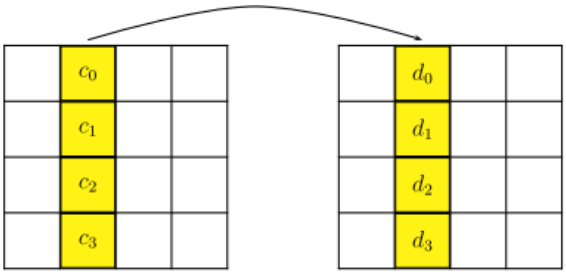


Figure 2.10 : *Transformation MixColums*

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} \quad (2.15)$$

Là encore la matrice utilisée est circulante. La transformation correspond en fait à une multiplication par un polynôme fixe $A(X) = 03.X^3 + 01.X^2 + 01.X + 02$**,** modulo $1 + X^4$:

$$\mathbf{d(X) = A(X) \times c(x) \mod X^4 + 1} \quad (2.16)$$

où $c(X) = c_0 + c_1 X + c_2 X^2 + c_3 X^3$, et $d(X) = d_0 + d_1X + d_2X^2 + d_3X^3$.

Le polynôme A(X) est premier avec $X^4 + 1$, il est donc inversible modulo $X^4 + 1$ et son inverse est $B(X) = 0B.X^3 + 0D.X^2 + 09.X + 0E$**.**

On retrouve donc **c(X)** à partir de **d(X)** en effectuant le produit $c(X) = B(X) \times d(X) \mod X^4 + 1$, ou en effectuant le produit matriciel

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \times \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} \quad (2.17)$$

2.3.3.7 La procédure AddRoundKey

La procédure *AddRoundKey* est très simple. Elle consiste à faire un ou exclusif entre les 128 bits de l'état St et les 128 bits de la clé de tour T. On obtient une nouvelle valeur de l'état.

$$St = St \oplus T \quad (2.18)$$

2.3.3.8 La procédure KeyExpansion

La clé de chiffrement K stockée dans un tableau de 4 lignes et N_k colonnes (N_k = 4, 6, 8) est

étendue en un tableau W ayant 4 lignes et $4 * n_r + 1$ colonnes. La clé de tour TK[i] ($0 \leq i \leq n_r$) est donnée par les 4 colonnes $4 * i$, $4 * i + 1$, $4 * i + 2$, $4 * i + 3$ du tableau W.

Il y a deux façons de construire le tableau W suivant que $N_k = 4$, 6 ou $N_k = 8$. La procédure de construction est nommée *ExpandedKey*.

1er cas : $N_k = 4$ ou $N_k = 6$

Entrée : la clé K (sous forme de tableau)

Sortie : le tableau W

ExpandedKey (K, W)

Début

Pour (j = 0 ; j < N_k ; j++)

Pour (i = 0 ; i < 4 ; i++) W [i, j] = K[i, j];

Pour (j = N_k ; j < $4(n_r + 1)$; j++)

Si (j mod N_k == 0)

Alors

$$W[0, j] = W[0, j - N_k] \oplus s(W[1, j-1]) \oplus RC[j / N_k];$$

Pour (i = 1 ; i < 4 ; i++)

$$W[i, j] = W[i, j - N_k] \oplus s(W[i+1 \bmod 4, j-1]);$$

Sinon

Pour (i = 0 ; i < 4 ; i++)

$$W[i, j] = W[i, j - N_k] \oplus s(W[i, j-1]);$$

Fin si ;

Fin

La procédure utilise la fonction **s** sur les octets définie précédemment. Elle utilise aussi des constantes de F_{256}, données par :

$$RC[i] = \alpha^i \tag{2.19}$$

où α est l'élément de F_{256} correspondant au polynôme **X (α = 02).** L'élévation à la puissance **i** se fait dans le corps F_{256}.

2ème cas : $N_k = 8$

Entrée : la clé *K* (sous forme de tableau)

Sortie : le tableau *W*

ExpandedKey (K, W)

Début

Pour (j = 0 ; j < N_k ; j++)

Pour (i = 0; i <4 ; i++) W [i, j] = K[i, j] ;

Pour (j = N_k ; j < 4(n_r + 1) ; j++)

Si (j mod N_k == 0)

Alors

$$W[0, j] = W[0, j - N_k] \oplus s(W[1, j-1]) \oplus RC[j / N_k];$$

Pour (i = 1 ; i < 4 ; i++)

$$W[i, j] = W[i, j - N_k] \oplus s(W[i+1 \bmod 4, j-1]);$$

Sinon si (j mod N_k == 4)

Pour (i = 0 ; i < 4 ; i++)

$$W[i, j] = W[i, j - N_k] \oplus s(W[i, j-1]);$$

Sinon

Pour (i = 0 ; i < 4 ; i++)

$$W[i, j] = W[i, j - N_k] \oplus W[i, j-1];$$

Fin si ;

Fin

2.4 Conclusion

Pour permettre de garantir une confidentialité maximum, les données sont tout d'abord chiffrées avant d'être dissimulées à l'aide d'un processus stéganographique. Nous avons vu dans ce chapitre les principales techniques de stéganograhie d'images que nous utiliserons, à savoir la technique LSB dans le domaine spatiale, et l'algorithme F5 dans le domaine fréquentiel, mais aussi le standard de chiffrement AES, afin de protéger au maximum l'information. Le chapitre suivant se consacrera sur une comparaison des meilleurs systèmes d'exploitation mobiles, afin de pouvoir choisir pour lequel programmer l'application de stéganographie.

CHAPITRE 3
SYSTEMES D'EXPLOITATION POUR MOBILE

3.1 Introduction

L'intérêt de ce chapitre est de pouvoir choisir pour quel système d'exploitation mobile programmer l'application. Pour ce faire, nous allons voir une comparaison des meilleurs systèmes d'exploitation mobiles. Nous les détaillerons ensuite afin de voir plus précisément leurs avantages et inconvénients, non seulement d'un point de vue utilisateur, mais aussi en tant que programmeur.

3.2 Les smartphones

Les Smartphones sont des appareils extrêmement sophistiqués, qui fournissent des fonctionnalités en plus de celles des téléphones mobiles classiques comme la télévision, la navigation sur le web, la consultation et l'envoi de courriers électroniques, la messagerie vocale et visuelle, etc. Dernièrement, on a aussi vu les Smartphones sophistiqués bénéficiant rapidement de la reconnaissance et synthèse vocale. Les Smartphones exécutent tous divers logiciels et applications grâce à des systèmes d'exploitation spécialement conçus pour les mobiles. Les Smartphones peuvent être personnalisés en y installant des applications additionnelles telles que des jeux ou des utilitaires grâce aux magasins d'applications en ligne (stores).

Le premier Smartphone, l'IBM Simon, fut conçu en 1992. Il a été commercialisé en août 1994. De nouvelles sociétés spécialisées dans les Smartphones, comme Research In Motion (avec le BlackBerry) se sont introduites parmi les principaux fabricants de téléphones classiques (comme Samsung, Nokia, LG) qui avaient déjà pris l'initiative de se lancer dans l'aventure. A partir de la fin de 2007, le marché des Smartphones s'étend considérablement jusqu'à dépasser en quelque années celui des téléphones mobiles classiques.

Un système d'exploitation mobile est un système d'exploitation conçu pour fonctionner sur un appareil mobile. Ce type de système d'exploitation se concentre entre autres sur la gestion de la connectivité sans fil et celle des différents types d'interface.

Il existe plusieurs systèmes d'exploitation spécifiques aux Smartphones. Selon l'article publié le 13 février 2015 par Wikipédia, les systèmes d'exploitation les plus utilisés parmi les autres sont Android, iOS (iPhone OS) et Windows Phone. L'usage des systèmes Android et iOS était

dominant, et constitue 96.5% en part de marché mondial, avec une part considérable d'Android de 84.6% [15].

	Android	**iOS**	**Windows Phone**	**BlackBerry OS**
Compagnie	Open Handset Alliance/Google	Apple, Inc	Microsoft	BlackBerry Ltd.
Part de marché	84.6%	11.9%	2.7%	0.6%
Dernière Version	5.0.2 (19 décembre 2014)	8.1.3 (17 Novembre 2014)	8.1.14147 (04 Août 2014)	10.2.1.3247 (25 Juin 2014)
Famille du système d'exploitation	Linux	Darwin	Windows NT 8+	QNX
Architecture de CPU supportée	ARM, x86, MIPS et variantes 64-bits des trois	ARM, ARM64	ARM	ARM
Appareils compatibles	HTC, Samsung Galaxy, Motorola,…	iPhone, iPod, iPad	Windows Phone, Nokia lumia 710…	BlackBerry torch, bold, curve...
Place de marché	Google Play	App Store	Windows Phone Store	BlackBerry World
Langage de développement	Java	Objective-C	Visual Basic / Visual C#	Java
Environnement compatible	Linux, Mac OS X et Windows	Mac OS X	Windows	Windows, Mac OS X, Linux
Prix du SDK	Gratuit	Gratuit	Gratuit	Gratuit
Coût de publication d'application dans l'application store officiel	25$ US pour une publication à vie sur Google Play	99$ US /an	Individuel : 19$ US/an Société : 99$ US/an	Gratuit

Tableau 3.01 : *Tableau de comparaison entre Android, iOS, Windows Phone et BlackBerry OS*

D’autres systèmes d’exploitation existent aussi comme : Symbian OS (Operating System) de Nokia, Linux, Palm webOS développé par Palm, puis HP (Hewlett-Packard Company), Bada de Samsung, MeeGo développé par Intel et Nokia.

3.2.1 iOS

iOS, anciennement iPhone OS, est le système d'exploitation mobile développé par Apple pour l'iPhone, l'iPod touch, l'iPad et l'Apple TV. Il est dérivé de OS X dont il partage les fondations (le noyau hybride XNU basé sur le micro-noyau Mach, les services Unix et Cocoa, etc.). iOS comporte quatre couches d'abstraction, similaires à celles de Mac OS X : une couche « Core OS », une couche « Core Services », une couche « Media » et une couche « Cocoa ». Le système d'exploitation occupe au maximum 3 Go de la capacité mémoire totale de l'appareil, selon l'appareil [16].

Ce système d'exploitation n'avait aucun nom officiel avant la publication du kit de développement iPhone (SDK ou Software Development Kit) le 6 mars 2008. Jusqu'à cette date, Apple se contentait de mentionner que « l'iPhone tourne sous OS X », une référence ambiguë au système d'exploitation source d'iOS, OS X. Ce n'est qu'à cette occasion que Scott Forstall présenta l'architecture interne du système d'exploitation, et dévoila alors le nom d'iPhone OS. Ce nom a été changé le 7 juin 2010 pour iOS. La marque commerciale « IOS » était utilisée par Cisco depuis plus de dix ans pour son propre système d'exploitation, IOS, utilisé sur ses routeurs. Pour éviter toute poursuite judiciaire, Apple a acquis auprès de Cisco une licence d'exploitation de la marque « IOS ».

Le kit de développement en question, disponible pour OS X, propose les outils nécessaires à la création d'une application pouvant tourner sous iOS. Si son téléchargement et son utilisation sont gratuits, la publication de telles applications requiert d'adhérer au programme des développeurs Apple, pour la somme de 99 $ par an. Il n'en demeure pas moins que cette offre peut s'avérer intéressante pour bon nombre de développeurs, étant donnée la taille du marché créé par iOS.

En effet, Apple a annoncé, lors d'un événement musical le 9 septembre 2009, avoir vendu 50 millions d'iPhones et d'iPod Touch. À titre informatif, les 40 millions d'appareils sous iOS n'avaient été dépassés que trois mois plus tôt, le 8 juin 2009. De surcroît, le portail App Store, dédié à l'exposition de toutes les applications développées pour ce système d'exploitation, est souvent présenté comme un modèle économique couronné de succès : avec un catalogue de 800

000 applications, qui ont fait l'objet de 50 milliards de téléchargements, l'App Store s'est imposé en 18 mois comme une référence parmi les kiosques d'applications mobiles.

iOS 8 est présenté lors de la WWDC 2014 (Worldwide Developers Conference), conjointement avec OS X Yosemite avec lequel une grande compatibilité est possible. Dans ce même événement, Apple annonce que 1 200 000 applications sont disponibles sur l'App Store et que 75 milliards d'applications ont été téléchargées [16].

Version d'iOS	Date de publication	Quelques évolutions
iOS 1.0	06-03-2007	La première version du système d'exploitation mobile d'Apple.
iOS 2.0	11-07-2008	Introduction d'un magasin d'applications tierce, l'App Store
iOS 3.0	17-06-2009	Nouvelle application, Dictaphone, permettant l'enregistrement des fichiers son.
iOS 4.0	21-06-2010	Introduction de multitâche et Facetime.
iOS 5.0	06-06-2011	Intégration d'iMessage et de Kiosque.
iOS 6.0	11-06-2012	Support de FaceTime sur les networks mobiles, nouveau privacycontrols.
iOS 7.0	10-06-2013	Ajout de Control Center, Touch ID scanneur
iOS 8.0	02-06-2014	clavier prédictif QuickType, iCloud Drive, installation obligatoire de l'Apps iBook, Podcast et de « aide »

Tableau 3.02 : *Quelques évolutions d'iOS*

La plupart des outils de développement du SDK étaient déjà présents dans Mac OS avant son arrivée. Cependant, ils gèrent désormais l'utilisation de l'iPhone, en tant que plate-forme de développement [16]:

- **Xcode** est l'Environnement de Développement Intégré par défaut sur OS X. Il permet l'écriture, la gestion et la compilation de projets de développement, écrits notamment en Objective-C. L'iPhone SDK y ajoute les bibliothèques de développement pour iOS. Il est donc possible pour le développeur de créer des projets d'applications pour ce système. Pour tester l'application, deux possibilités existent : le développeur peut brancher un iPhone ou iPod

Touch à son ordinateur Mac, puis y lancer l'application comme test à condition d'adhérer au programme des développeurs d'Apple, ou lancer l'application en test dans iPhone Simulator.

- **Interface Builder** permet de construire une interface pour Cocoa Touch manuellement, à l'aide de glisser-déposer. Il permet également de traduire facilement une application dans plusieurs langues. De plus, il permet de gérer visuellement le schéma Modèle-Vue-Contrôleur, en connectant des éléments d'une interface à un code écrit pour eux au préalable, à l'aide d'un glisser-déposer. Finalement, le fichier d'interface ainsi créé est ajouté au projet Xcode.
- **Instruments** est un outil de monitoring informatique. Il permet, une fois l'application lancée sur un iPhone ou iPod Touch branché à l'ordinateur, d'observer en temps réel ses performances au niveau du processeur, mais également, par exemple, du moteur graphique ou de l'accéléromètre. Par ailleurs, il est également possible de surveiller les performances système dans iPhone Simulator.
- **iPhone Simulator** (anciennement Aspen Simulator) est le seul de ces outils à avoir été développé spécifiquement pour l'iPhone SDK. Il simule de manière logicielle un iPhone virtuel, qui peut exécuter des applications directement sur l'ordinateur. Les mouvements Multitouch sont alors reproduits manuellement à la souris par l'utilisateur, et il est possible de faire pivoter le simulateur grâce à des raccourcis clavier. Par ailleurs, l'utilisateur est en mesure de choisir quel matériel et version du firmware il désire utiliser.
- **Swift** : lors de la WWDC 2014, Apple lève le voile sur un nouveau langage de programmation qui remplacera à terme *Objective-C*.

Figure 3.01 : *Logo du système d'exploitation mobile iOS*

iOS est un peu plus vieux qu'Android, d'une courte année. Mais c'est aussi celui à qui l'on doit l'idée du magasin d'application, avec le succès que l'on connait.

Les mises à jour sont beaucoup moins problématiques que sur Android : l'ensemble des téléphones reçoit les nouveautés au fur et à mesure de leur sortie. Dernièrement, la version iOS 8 a

été déployée, introduisant quelques changements dans les fonctionnalités, mais à l'ergonomie toujours très bonne. iOS dispose de l'interface le plus accessible, avec un système d'icône et de dossier aussi simple à prendre en main qu'efficace.

Figure 3.02 : *Interface d'iOS*

L'interface du système d'exploitation iOS est fondée sur le concept de la manipulation par contact tactile de l'écran, incluant notamment la technologie Multi-touch, permettant de reconnaître des gestes à plusieurs doigts simultanés, et un accéléromètre détectant les mouvements de l'appareil. Pour synchroniser son appareil et ajouter de la musique ou des vidéos, il est indispensable de passer par le logiciel iTunes, ce qui est assez contraignant. Globalement, iOS est plus fermé qu'Android : impossible d'installer un programme qui ne vient pas de l'App Store, à moins de « jailbreaker » le téléphone, une manipulation qui n'est pas à la portée de tout le monde et qui au passage fait perdre la garantie [16].

Le système iOS contient donc plusieurs limites imposées par Apple :

- le système est fermé (il est nécessaire de débrider le système pour bénéficier de certaines fonctionnalités non proposées par Apple : c'est ce qu'on appelle le *jailbreak*) ;
- le format Flash d'Adobe n'est pas supporté et d'après Apple ne le sera jamais (principalement pour allonger la durée de vie de la batterie et ne pas ralentir le système), de nombreux sites de vidéo ont remplacé leur format de vidéo par les formats H.264 et HTML5 spécialement pour l'iPad ou l'iPhone ;

- le système d'exploitation est destiné aux terminaux Apple, contrairement à d'autres systèmes d'exploitation mobiles comme Android, MeeGo, ou Windows Phone.
- l'utilisateur a l'obligation de passer par le magasin officiel pour installer un logiciel. Apple a rédigé et plusieurs fois modifié les conditions générales de son magasin, de façon à écarter la concurrence pour certains logiciels applicatifs, comme le navigateur Web par exemple.

3.2.2 Windows Phone

Windows Phone est un système d'exploitation mobile développé par Microsoft pour succéder à Windows Mobile, sa précédente plateforme logicielle qui a été renommée pour l'occasion en Windows Phone Classic. Contrairement au système qu'il remplace, Windows Phone est d'abord principalement destiné au grand public plutôt qu'au marché des entreprises. Cependant depuis Windows Phone 8, Microsoft propose des fonctions avancées pour les entreprises en offrant, par exemple, un espace d'applications réservé aux entreprises. Il a été lancé le 21 octobre 2010 en Europe, à Singapour, en Australie et en Nouvelle-Zélande, le 8 novembre 2010 aux États-Unis et au Canada, puis le 24 novembre 2010 au Mexique.

Selon Microsoft, le développement du système d'exploitation est parti d'une feuille blanche, et s'est terminé entre fin août et début septembre 2010 avec notamment la disponibilité du kit de développement final et la diffusion d'un kit de présentation aux constructeurs. Contrairement à Windows Mobile, l'interface homme-machine de Windows Phone repose nativement sur l'utilisation d'un écran tactile multipoints. Avec Windows Phone, Microsoft propose une interface utilisateur dénommée « Modern UI » avec un système de tuiles dynamiques, très différente de ce que l'on peut avoir l'habitude avec iOS ou Android. Windows Phone devient en mai 2013, le 3ème système d'exploitation mobile [17].

Le système d'exploitation était initialement disponible en cinq langues : anglais, allemand, espagnol, français et italien, cependant la mise à jour nommée Mango, sortie le 27 septembre 2011, étend le nombre de langues supportées à 25. Windows Phone Store, la boutique en ligne de Microsoft dédiée au téléchargement des applications pour Windows Phone, propose plus de 200 000 applications en décembre 2013. Windows Phone fait l'objet de mises à jour annuelles, la prochaine version aura pour nom Windows 10. En effet, les noms "Windows Phone" et "Windows RT" vont disparaître et seront remplacés par "Windows" car Microsoft travaille pour unifier ses Systèmes d’exploitation.

De même façon qu'Apple avec iOS, Windows Phone sera mis à jour de façon automatique sans passer par les constructeurs afin d'offrir une expérience utilisateur identique entre tous les modèles de téléphones et éviter la fragmentation des versions comme sur la plateforme Android. Les mises à jour passeront uniquement par le Zune Software pour les mises à jour importantes ou bien sur le téléphone même over-the-air pour les mises à jour mineures. Microsoft a annoncé que dans un futur proche, toutes les mises à jour qu'elles soient importantes ou mineures se dérouleraient « over-the-air ». Les applications téléchargées peuvent être mises à jour via le Store.

Microsoft a l'intention de diffuser des mises à jour mineures qui proposent de nouvelles fonctionnalités plusieurs fois par an avec une mise à jour plus importante chaque année.

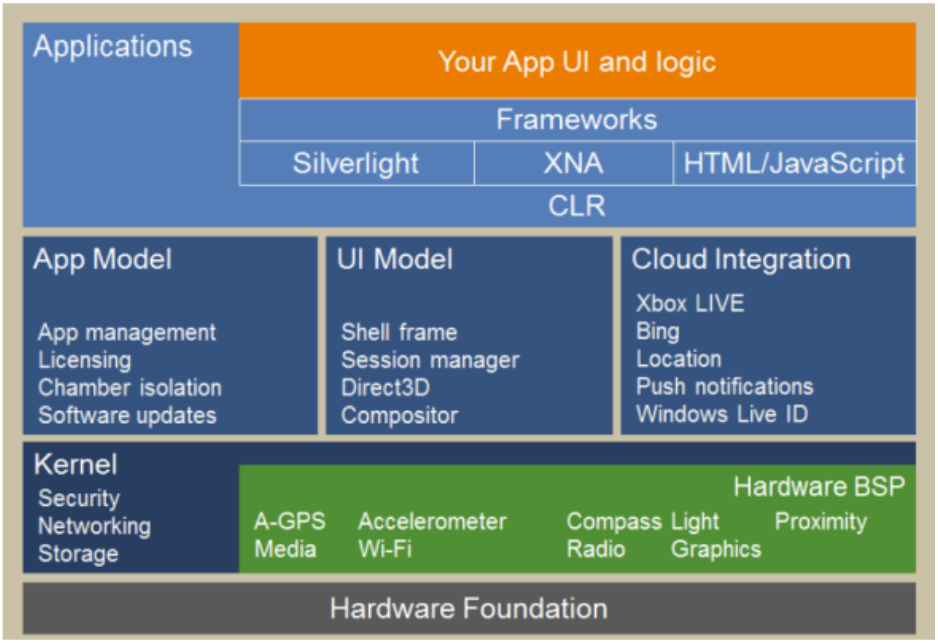


Figure 3.03 : *Architecture de Windows Phone 7*

Les outils nécessaires pour pouvoir réaliser des applications pour Windows Phone sont :

- Windows Phone Developper Tools
- Visual Studio 2010 Express for Windows Phone
- XNA Game Studio4.0
- On-Screen Windows Emulateur.

Ces outils sont disponibles sur le site ‹‹ http://developer.windowsphone.com ››.

Les applications sont écrits avec langage de programmation C#. Pour rendre les applications développées sur cette plateforme installables sur un Windows Phone actuel, il faut s'inscrire sur le site ‹‹ http://developer.windowsphone.com ››.

Figure 3.04 : *Windows phone (Operating System)*

Figure 3.05 : *Interface de Windows Phone 8.1*

De l'eau a coulé sous les ponts depuis Windows Mobile. Windows Phone est une refonte totale à tout point de vue. L'interface partagée avec Windows 8 est à base de tuiles, ce qui rend le système très sobre et lisible. Et avec la version 8.1 de Windows Phone, la personnalisation est au rendez-vous : on peut désormais appliquer un fond d'écran sur les tuiles, les regrouper dans des « dossiers », etc. La taille des icônes peut être ajustée et certaines sont dynamiques, ce qui donne encore plus de vie à l'ensemble. Enfin, le système d'exploitation mobile profite d'un centre de notifications très complet, qui permet de consulter les dernières alertes, de gérer l'affichage ou d'activer la connexion Wi-Fi.

Comme nous venons de le voir, la dernière version du système d'exploitation mobile de Microsoft est Windows Phone 8.1. Son magasin d'applications se remplit progressivement, plus de 250 000 aux dernières nouvelles. Pendant longtemps, il a souffert de l'absence de certains gros titres. Heureusement, la plupart finissent par être portées (Instagram, Vine ainsi que Waze de Google sont désormais présents sur Windows Phone).

Tout comme avec Android, il est possible d'ajouter des fichiers très simplement sur la mémoire interne du téléphone, mais ce n'est pas le cas des programmes, de la même manière qu'iOS.

3.2.3 BlackBerry OS

BlackBerry est une ligne de téléphones intelligents développée depuis 1999, d'abord sous le nom de RIM (Research In Motion), nom de produit BlackBerry. En janvier 2013, elle a adopté le nom unique de BlackBerry. Ces téléphones sont fabriqués par la compagnie canadienne Research In Motion (RIM), utilisant le système d'exploitation propriétaire Blackberry OS.

La technique BlackBerry, permettant à l'origine de recevoir et envoyer des courriels, est utilisée pour recevoir tous types de notifications en mode push, c'est-à-dire en temps réel, sans avoir besoin d'aller se connecter à un serveur, en étant constamment connecté sur le réseau Blackberry : pour les e-mails sur 1 à 10 comptes de messagerie différents, mais aussi pour la messagerie instantanée BlackBerry Messenger (BBM) ou les notifications des réseaux sociaux comme Facebook ou Twitter. Le mode de compression réduit la taille du mail, ce qui facilite la synchronisation de ses courriels avec le serveur de messagerie électronique via le réseau de téléphonie mobile sur lequel l'appareil est connecté (GSM ou Global System for Mobile Communications, GPRS ou General Packet Radio Service, UMTS ou Universal Mobile Telecommunications System, etc.). Ainsi, envoyer un courriel est aussi simple que d'envoyer un SMS (Short Message Service) ou un MMS (Multimedia Messaging Service).

Le BlackBerry permet la lecture des pièces jointes aux formats « .ppt », « .pdf », « .wpd », « .html », « .txt », « .zip », « .jpg », « .bmp », « .png », « .gif » et « .tiff ». Deux méthodes de lecture sont possibles [18] :

- présentation de la table des matières du document sur le BlackBerry : chargement au choix de la partie désirée ;
- chargement du contenu au format texte et image pour optimiser l'affichage et le temps de chargement.

L'avantage de cette technique est qu'elle permet de lire rapidement les pièces jointes de grande taille (doc, xls, ppt, pdf, txt, images, etc.) grâce à la compression de l'information. Les pièces jointes peuvent voir leur taille divisée par 200. Par exemple, un courriel reçu avec une photo de 3 Mo sera transmis sur le BlackBerry avec la même image réduite à 20 Ko environ (soit une taille divisée par 150).

Cette technique permet également de naviguer sur Internet. L'interface Web permet d'accéder à ses comptes courriel BlackBerry et de les gérer (ajout, suppression, etc.)

Figure 3.06 : *BlackBerry (Operating System)*

Depuis l'arrivée des grands concurrents de RIM (iOS, Bada et Android), les plates-formes de programmation pour le BlackBerry se sont mises en place. Les applications du BlackBerry étant programmées en Java, RIM a proposé en téléchargement une bibliothèque de composants créés spécialement pour le développement d'applications sur la plateforme de développement Eclipse. La JavaDoc de l'API BlackBerry JDE 7.0.0 est disponible.

Les terminaux BlackBerry acceptent de nombreux langages de programmation pour développer des applications : Pour Blackberry OS 7 et antérieures, il est possible d'utiliser au choix du java ou du HTML5. Pour BlackBerry 10, le choix du langage s'élargit au C/C++, HTML5, Adobe AIR ou même Java Android. Ainsi, les applications Android sont compatibles avec BlackBerry 10.

Figure 3.07 : *Interface de BlackBerry 10*

Pour sortir de la tourmente, BlackBerry s'est donné du mal avec BlackBerry 10. Bien né, ce système d'exploitation mobile se veut ultra-connecté grâce au Hub, un volet qui conglomère toutes les notifications (réseaux sociaux, mails, SMS, etc). Il demande un petit temps d'adaptation, tout comme l'interface massivement multitâche aux raccourcis (glissement de doigt à certains endroits de l'écran) parfois peu évidents.

Le système regorge de bonnes idées à commencer par la possibilité de séparer son espace de travail de sa vie privée sur un seul et même téléphone sans sombrer dans la schizophrénie. Le magasin d'application maison, l'App World, continue à se remplir, mais sa force est sa compatibilité avec les programmes Android, qui ne nécessitent qu'une petite optimisation de la part des développeurs.

3.2.4 Android

3.2.4.1 La création d'Android

Penser à Android revient à penser immédiatement à Google, et pourtant il faut savoir que cette multinationale n'est pas à l'initiative du projet. D'ailleurs, elle n'est même pas la seule à contribuer à plein temps à son évolution. À l'origine, « Android » était le nom d'une PME (Petites et Moyennes Entreprises) américaine, créée en 2003 puis rachetée par Google en 2005, qui avait la ferme intention de s'introduire sur le marché des produits mobiles. La gageure, derrière Android, était de développer un système d'exploitation mobile plus intelligent, qui ne se contenterait pas uniquement de permettre d'envoyer des SMS et transmettre des appels, mais qui devait permettre à l'utilisateur d'interagir avec son environnement (notamment avec son emplacement géographique). C'est pourquoi, contrairement à une croyance populaire, il n'est pas possible de dire qu'Android est une réponse de Google à l'iPhone d'Apple, puisque l'existence de ce dernier n'a été révélée que deux années plus tard.

C'est en 2007 que la situation prit une autre tournure. À cette époque, chaque constructeur équipait son téléphone d'un système d'exploitation propriétaire. Chaque téléphone avait ainsi un système plus ou moins différent. Ce système entravait la possibilité de développer facilement des applications qui s'adapteraient à tous les téléphones, puisque la base était complètement différente. Un développeur était plutôt spécialisé dans un système particulier et il devait se contenter de langages de bas niveaux comme le C ou le C++. De plus, les constructeurs faisaient en sorte de livrer des bibliothèques de développement très réduites de manière à dissimuler leurs secrets de fabrication. En janvier 2007, Apple dévoilait l'iPhone, un téléphone tout simplement

révolutionnaire pour l'époque. L'annonce est un désastre pour les autres constructeurs, qui doivent s'aligner sur cette nouvelle concurrence. Le problème étant que pour atteindre le niveau d'iOS (iPhone OS), il aurait fallu des années de recherche et développement à chaque constructeur...

C'est pourquoi est créée en novembre de l'année 2007 l'Open Handset Alliance (OHA), et qui comptait à sa création 35 entreprises évoluant dans l'univers du mobile, dont Google. Cette alliance a pour but de développer un système open source (c'est-à-dire dont les sources sont disponibles librement sur internet) pour l'exploitation sur mobile et ainsi concurrencer les systèmes propriétaires, par exemple Windows Mobile et iOS. Cette alliance a pour logiciel vedette Android, mais il ne s'agit pas de sa seule activité [19].

Figure 3.08 : *Logo de l'Open Handset Alliance*

L'OHA compte à l'heure actuelle 80 membres. Android est à l'heure actuelle le système d'exploitation pour smartphones et tablettes le plus utilisé.

Les prévisions en ce qui concerne la distribution d'Android sur le marché sont très bonnes avec de plus en plus de machines qui s'équipent de ce système. Bientôt, il se trouvera dans certains téléviseurs (Google TV) et les voitures. Android sera partout.

Figure 3.09 : *Android OS*

3.2.4.2 Les versions d'Android

Les versions se succèdent rapidement et les changements qui les accompagnent sont souvent conséquents en termes de nouvelles fonctionnalités et d'améliorations.

Version Android	Nom de la version	Date de release	Quelques évolutions
5.0	Lollipop	03-11-2014	Nouveau design de l'interface graphique : Material Design, Nouveau moteur d'exécution ART, Projet Volta (optimiser la consommation d'énergie et gagner en autonomie), Amélioration du système de notifications, Activation par défaut du chiffrement des données utilisateur Disponibilité d'Android TV et Android Auto
4.4	KitKat	14-04-2014	Interface translucide, Framework pour imprimer, Framework pour la gestion des fichiers.
4.1	Jelly Bean	09-07-2012	Assistance vocale, accessibilité : mode gestuel 'Braille', WIFI-Direct service discovery, vsync timing
4.0	IceCream Sandwich	19-10-2011	WI-FI direct, Bluetooth Health Devie profile, Control over network data, Grid Layout.
3.2	Honeycomb	15-07-2011	Support des processeurs Qualcomm, Support des tablettes tactiles de 7 pouces
2.3	Gingerbread	06-12-2010	Support de la VoIP et SIP. Gestionnaire de téléchargement, support de plusieurs cameras.
2.2	Froyo	20-05-2010	Implémentation de JIT, partage de connexion USB.
2.0	Eclair	26-10-2009	Bluetooth, support de plus de taille d'écran.
1.6	Donut	15-09-2009	Google navigation (GPS gratuit)
1.5	Cupcake	30-04-2009	Envoi de vidéos vers YouTube et Picasa, rotation automatique
1.1	Banana bread	09-02-2009	Support pour sauvegarder les fichiers attachent aux MMS.
1.0	Apple pie	23-09-2008	Début de l'aventure Android.

Tableau 3.03 : *Quelques évolutions des versions d'Android*

Figure 3.10 : *Android OS*

L'interface est personnalisable à l'envie, à l'aide de widget ou de raccourcis. Mais parfois, l'embarras du choix peut perdre l'utilisateur à cause d'une ergonomie pas toujours idéale. Les bidouilleurs peuvent quant à eux aller plus loin en modifiant le système afin d'optimiser la rapidité ou implanter des fonctions. La gestion de la mémoire sur un téléphone Android est très simple : il suffit de le brancher à un ordinateur pour y accéder. Il y est possible d'ajouter n'importe quel type de documents et surtout, d'installer des applications à l'aide de fichiers .apk.

3.2.4.3 Architecture d'Android

L'environnement de développement est basé sur une architecture reposant autour d'un noyau Linux. La plateforme Android est composée de cinq couches principales :

- Un noyau Linux qui lui confère des caractéristiques multitâches.
- Des bibliothèques graphiques, multimédias.
- La Dalvik Virtuel Machine, une machine virtuelle adaptée pour java.
- Une plateforme applicative pour la gestion des fenêtres, du contenu, de la téléphonie, etc.
- Des applications.

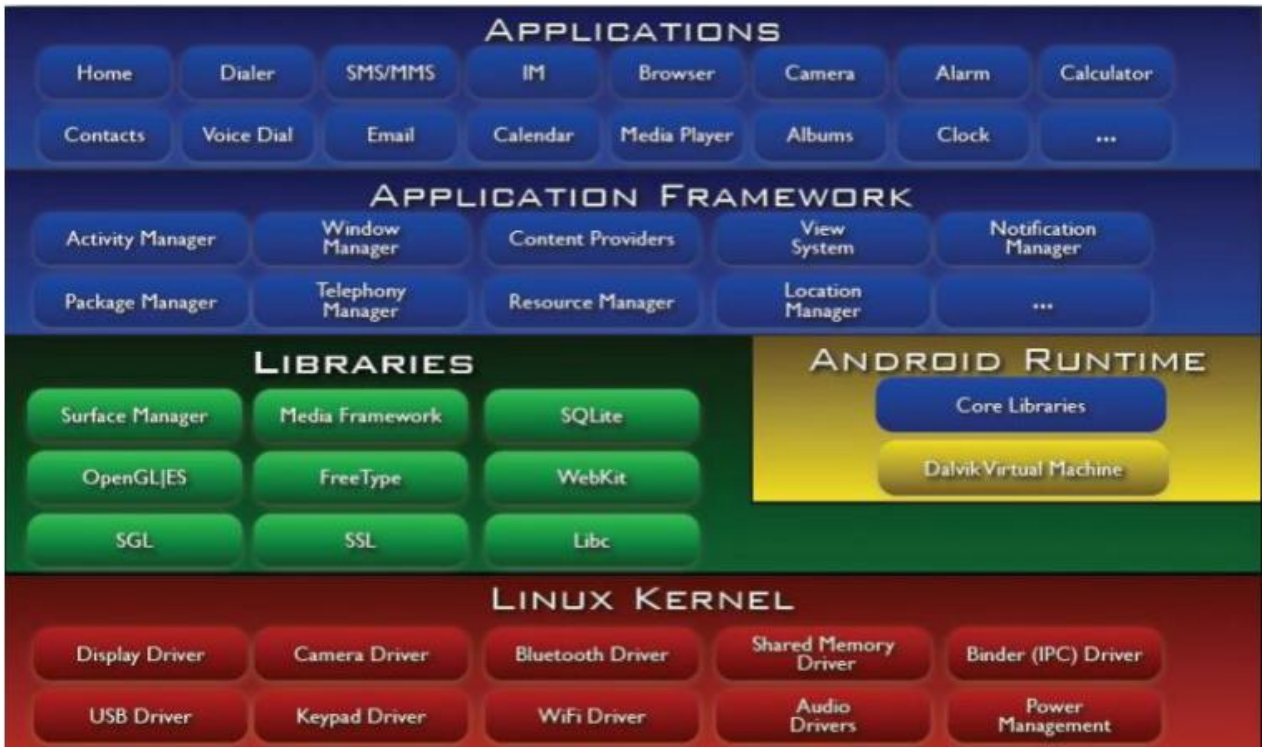


Figure 3.11 : *L'architecture de la plateforme Android*

3.2.4.4 La philosophie et les avantages d'Android

Open source

Le contrat de licence pour Android respecte les principes de l'open source, c'est-à-dire que nous pouvons à tout moment télécharger les sources et les modifier selon nos goûts. Noter au passage qu'Android utilise des bibliothèques open source puissantes, comme par exemple SQLite pour les bases de données et OpenGL pour la gestion d'images 2D et 3D.

Gratuit (ou presque)

Android est gratuit, autant pour nous que pour les constructeurs. En revanche, pour poster vos applications sur le Play Store, il en coûtera la modique somme de 25$. Ces 25$ permettent de publier autant d'applications que nous le souhaitons, à vie.

Facile à développer

Toutes les API (Application Programming Interface) mises à disposition facilitent et accélèrent grandement le travail. Ces APIs sont très complètes et très faciles d'accès. Une API, ou « interface de programmation » en français, est un ensemble de règles à suivre pour pouvoir dialoguer avec d'autres applications. Dans le cas de Google API, il permet en particulier de communiquer avec Google Maps.

Facile à vendre

Le Play Store (anciennement Android Market) est une plateforme immense et très visitée ; c'est donc une mine d'opportunités pour quiconque possède une idée originale ou utile.

Flexible

Le système est extrêmement portable, il s'adapte à beaucoup de structures différentes. Les smartphones, les tablettes, la présence ou l'absence de clavier ou de trackball, différents processeurs... On trouve même des fours à micro-ondes qui fonctionnent à l'aide d'Android.

Non seulement c'est une immense chance d'avoir autant d'opportunités, mais en plus Android est construit de manière à faciliter le développement et la distribution en fonction des composants en présence dans le terminal (si notre application nécessite d'utiliser le Bluetooth, seuls les terminaux équipés de Bluetooth pourront la voir sur le Play Store).

Ingénieux

L'architecture d'Android est inspirée par les applications composites, et encourage par ailleurs leur développement. Ces applications se trouvent essentiellement sur internet et leur principe est que nous pouvons combiner plusieurs composants totalement différents pour obtenir un résultat surpuissant.

3.3 Parts de marché

Selon l'article du site www.iwebyou.fr/actualites/statistiques-internet-janvier-2015, publié par Joel Garcin le 26 janvier 2015, sur l'année 2014 [20], la fréquentation des sites Internet est en hausse depuis les smartphones, mais stagne sur les tablettes. L'article présente l'évolution sur 1 an des technologies utilisées, navigateurs et systèmes d'exploitation en France et dans le Monde.

Les mobiles en forte hausse, faible avancée des tablettes : sur 2014, le pourcentage d'utilisateurs de terminaux mobiles (Smartphones et Tablettes) sur Internet est passé de 28% à 37% soit plus d'1 utilisateur sur 3. C'est la navigation par smartphones (hors applications) qui a le plus augmentée, passant de 22% à 31% des utilisations [20].

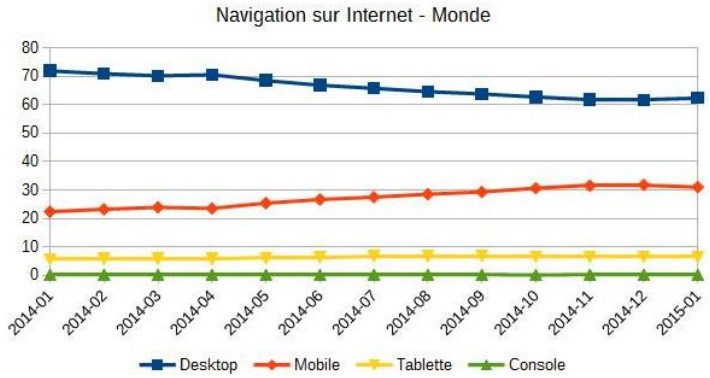


Figure 3.12 : *Navigation sur Internet dans le monde selon www.iwebyou.fr*

En France, cette utilisation reste très en retard sur la moyenne mondiale, mais décolle vraiment depuis Juillet 2014. En revanche, les internautes français utilisent plus fréquemment une tablette pour aller sur Internet que la moyenne mondiale (8% contre 6.5%) [20].

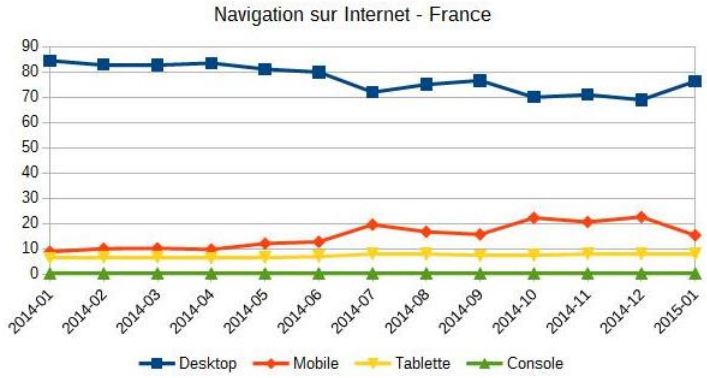


Figure 3.13 : *Navigation sur Internet en France www.iwebyou.fr*

Android sur smartphone confirme son avance sur iOS : Android domine le marché et distance de plus en plus iOS. L'arrivée fin septembre de iOS 8 et des iPhones 6 ne semble pas rattraper les places perdues au profit de Samsung et HTC (High Tech Computer Corporation) [20].

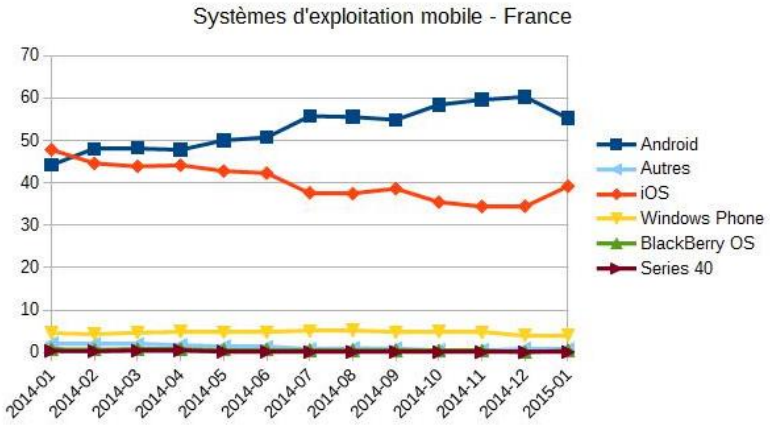


Figure 3.14 : *Utilisation des systèmes d'exploitation mobile en France www.iwebyou.fr*

Cette large avance est encore plus criante sur le marché mondial, où iOS ne représente que 23% contre 55% pour les mobiles équipés d'Android.

Le prix qui jusque-là s'expliquait par une avance technologique, certaine joue en la défaveur de l'iPhone, qui n'apporte plus de réelle innovation depuis l'iPhone 4 alors qu'Android accroit son avance technologique sur des mobiles largement moins chers. L'arrivée de nouveaux constructeurs utilisant Android comme Lenovo et Oneplus va très certainement profiter à Android [20].

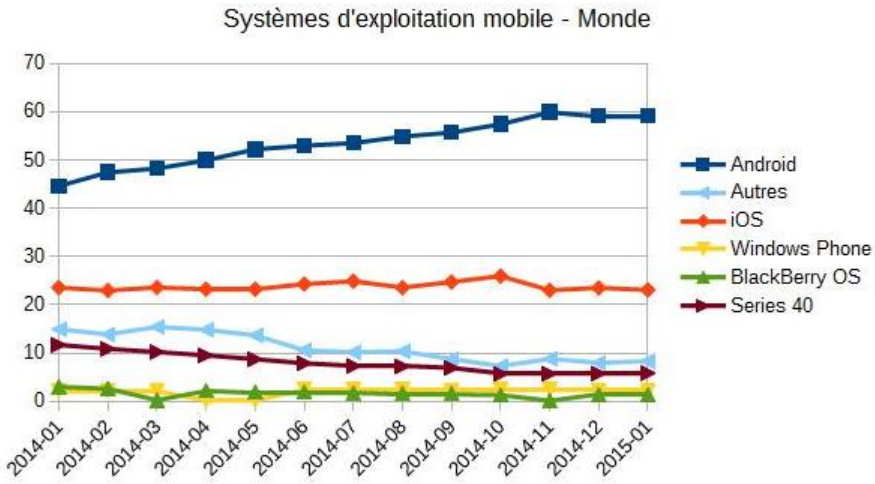


Figure 3.15 : *Utilisation des systèmes d'exploitation mobile dans le Monde www.iwebyou.fr*

Les tablettes toujours dominées par Ipad : En revanche, du côté des tablettes, iOS domine encore largement en France avec ses iPads mais la baisse se confirme au profit d'Android et plus modestement de Windows 8.1 RT [20].

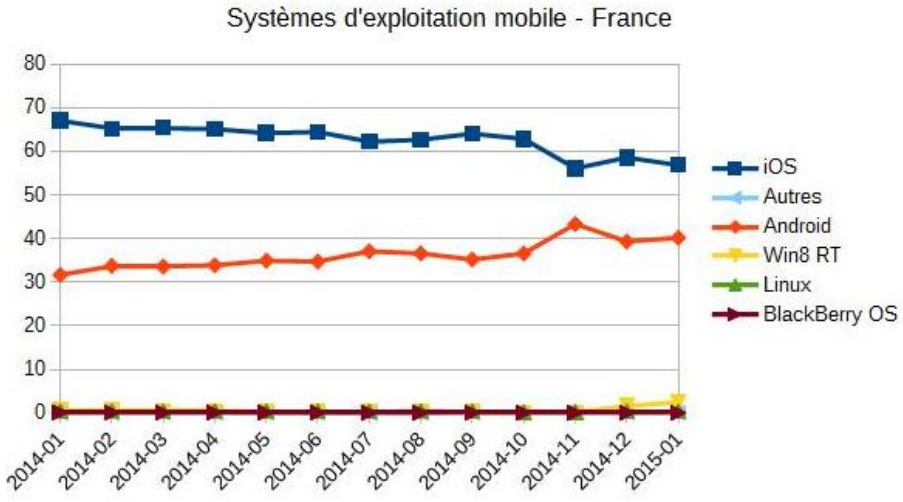


Figure 3.16 : *Utilisation des tablettes en France www.iwebyou.fr*

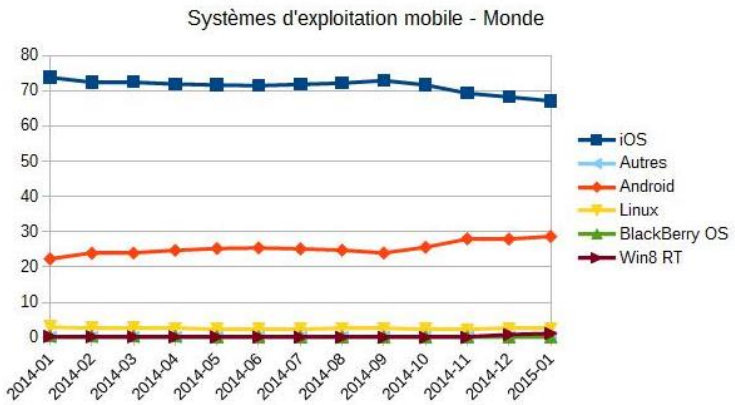


Figure 3.17 : *Utilisation des tablettes dans le monde www.iwebyou.fr*

En outre, selon l'article publié par ***Wikipedia***, mis à jour le 13 Fevrier 2015, qui fournit une comparaison entre plusieurs systèmes d'exploitation mobile, Google et Apple mènent la danse en termes de parts de marché avec respectivement 84,6 % et 11,9 % [15].

Selon l'article publié par ***l'IDC (International Data Corporation) Worldwide Quarterly Mobile Phone Tracker,*** le 24 Février 2015, la domination d'Android sur le marché mondial des smartphones est indiscutable. Sur l'ensemble de l'année, 1,059 milliards de smartphones Android ont été livrés dans le monde par les constructeurs. Non seulement ces livraisons ont progressé de 32%, mais surtout elles représentent 81,5% des smartphones écoulés sur la planète en 2014. Avec iOS, les deux OS mobiles ont représenté pas moins de 96,3% de l'ensemble des smartphones écoulés dans le monde en 2014. C'est plus encore qu'en 2013 (93,8%). Quant aux autres, ils se livrent d'abord une bataille d'arrière-garde [21].

Système d'exploitation mobile	**Livraisons 2014 (millions)**	**Part de marché 2014**	**Livraisons 2013 (millions)**	**Part de marché 2013**	**Evolution année sur année**
Android	1,059.3	81.5%	802.2	78.7%	32.0%
iOS	192.7	14.8%	153.4	15.1%	25.6%
Windows Phone	34.9	2.7%	33.5	3.3%	4.2%
BlackBerry	5.8	0.4%	19.2	1.9%	-69.8%
Autres	7.7	0.6%	2.3	0.2%	234.8%
Total	1,300.4	100.0%	1,018.7	100.0%	27.7%

Tableau 3.04 : *Statistiques d'évolutions des systèmes d'exploitation mobile selon l'IDC*

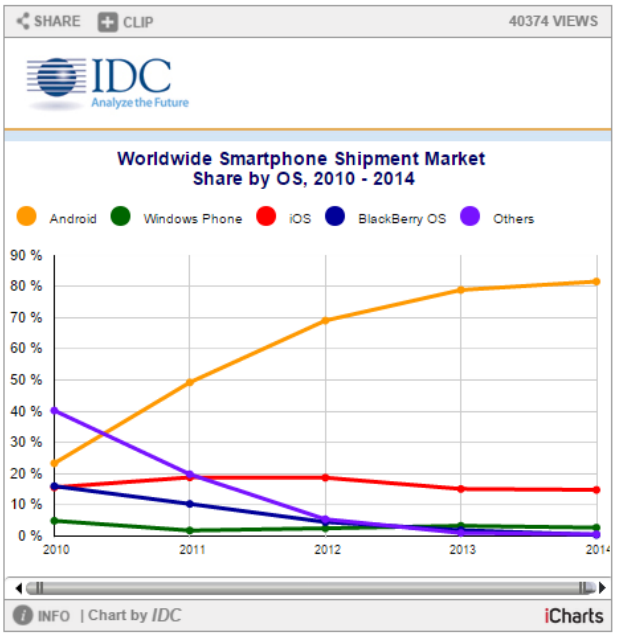


Figure 3.18 : *Part de marché mondial des systèmes d'exploitation mobile selon l'IDC*

Android bénéficie de l'entrée de gamme. Pour autant, et malgré une avance confortable, l'écosystème Android connaît quelques évolutions. En raison du fort ralentissement de Samsung en 2014, la croissance des livraisons de smartphones repose désormais plus sur des constructeurs de taille moindre.

Derrière Android et iOS, seul Windows Phone semble encore pouvoir exister. « Plutôt qu'une bataille pour le 3ème écosystème derrière Android et iOS, 2014 a donné lieu à des escarmouches, avec Windows Phone devançant BlackBerry, Firefox, Sailfish et le reste, mais sans qu'aucune de ces plateformes n'enregistrent les progrès nécessaires pour défier les deux premiers » commente l'analyste d'IDC, Melissa Chau.

3.4 Conclusion

Dans ce chapitre, nous avons vu une comparaison des meilleurs systèmes d'exploitation mobiles : Android, iOS, Windows Phone et BlackBerry OS. Android détient à lui seul plus de 80% de la part de marché mondial, suivi de iOS d'environ 14% durant l'année 2014. Cette popularité est surement due à la philosophie d'Android qui se veut d'être Open Source, gratuit ou presque, facile à développer, facile à vendre, flexible, et ingénieux. C'est ainsi donc, que nous choisissons de développer l'application de stéganographie pour la plateforme Android. Le chapitre suivant concerne justement la conception et le développement de l'application.

CHAPITRE 4
CONCEPTION ET DEVELOPPEMENT DE L'APPLICATION

4.1 Introduction

Ce chapitre est certainement un des plus importants de tous, étant donné qu'il concerne la conception et le développement de l'application proprement dite. C'est dans le prochain chapitre que nous décrirons les tests effectués et l'interprétation des résultats. Mais d'abord, certaines difficultés du développement pour des systèmes embarqués vont être abordées pour mieux cerner la délicatesse du projet, ainsi que quelques notions à savoir pour programmer sur Android, le système d'exploitation choisi pour diverses raisons expliquées dans le chapitre précédent.

4.2 Les difficultés du développement pour des systèmes embarqués

Il existe certaines contraintes pour le développement Android, qui ne s'appliquent pas au développement habituel, par exemple au niveau des mémoires vives RAM (Random Access Memory) des téléphones généralement un peu faibles par rapport aux ordinateurs.

Voici les principales contraintes à prendre en compte quand on développe pour un environnement mobile [19]:

- Il faut pouvoir interagir avec un système complet sans l'interrompre. Android entreprend des activités pendant que notre application est utilisée, il reçoit des SMS et des appels, entre autres. Il faut respecter une certaine priorité dans l'exécution des tâches, par exemple il serait futile de bloquer les appels de l'utilisateur pour qu'il puisse terminer d'utiliser l'application.
- Le système n'est généralement pas aussi puissant qu'un ordinateur classique, il faudra donc exploiter tous les outils fournis afin de débusquer les portions de codes qui nécessitent des optimisations.
- La taille de l'écran est réduite, et il existe par ailleurs plusieurs tailles et résolutions différentes. Notre interface graphique doit s'adapter à toutes les tailles et toutes les résolutions.
- En outre, les interfaces tactiles sont peu pratiques en cas d'utilisation avec un stylet et/ou peu précises en cas d'utilisation avec les doigts, d'où des contraintes liées à la programmation événementielle plus rigides. En effet, il est possible que l'utilisateur se trompe souvent de bouton. Très souvent s'il a de gros doigts.

- Enfin, en plus d'avoir une variété au niveau de la taille de l'écran, on a aussi une variété au niveau de la langue, des composants matériels présents et des versions d'Android. Il y a une variabilité entre chaque téléphone et même parfois entre certains téléphones identiques. C'est un travail en plus à prendre en compte.

Les conséquences de telles négligences peuvent être terribles pour l'utilisateur. Si nous saturons le processeur, il ne pourra plus rien faire excepté redémarrer. Faire crasher une application ne fera en général pas complètement crasher le système, cependant il pourrait bien s'interrompre quelques temps et irriter profondément l'utilisateur.

Le développement de programmes pour un téléphone portable est donc différent de l'écriture d'applications pour des machines de bureau, du développement de sites web ou de la création de programmes serveurs.

A cela, Android essaie de nous faciliter les choses :

- Il fournit un langage de programmation connu (Java), avec des bibliothèques relativement classiques (certaines API d'Apache, par exemple), ainsi qu'un support pour les outils auxquels nous sommes habitués (Eclipse, notamment).
- Il nous offre un framework suffisamment rigide et étanche pour que nos programmes s'exécutent "correctement" sur le téléphone, sans interférer avec les autres applications ou le système d'exploitation lui-même.

4.3 Outils de développement d'Android

De manière générale, n'importe quel matériel permet de développer sur Android que ce soit sur Windows, Mac OS X ou une distribution Linux. Pour un environnement Windows, sont tolérés XP (en version 32 bits), Vista (en version 32 et 64 bits) et 7 (aussi en 32 et 64 bits). Officieusement (en effet, Google n'a rien communiqué à ce sujet), Windows 8 est aussi supporté en 32 et 64 bits. Sous Mac, il faudra utiliser Mac OS 10.5.8 ou plus récent et un processeur x86. Sous GNU/Linux, Google conseille d'utiliser une distribution Ubuntu plus récente que la 10.04. Normalement, n'importe quelle distribution convient à partir du moment où la bibliothèque GNU C (glibc) est au moins à la version 2.7. Pour une distribution 64 bits, elle devra être capable de lancer des applications 32 bits [19].

Pour notre cas, nous avons utilisé Windows 8 64 bits.

4.3.1 Le Java Development Kit

Il existe deux plateformes en Java :

- Le **JRE** (Java Runtime Environment), qui contient la JVM (Java Virtual Machine), les bibliothèques de base du langage ainsi que tous les composants nécessaires au lancement d'applications ou d'applets Java. En gros, c'est l'ensemble d'outils permettant d'exécuter des applications Java.
- Le **JDK** (Java Development Kit), qui contient le JRE (afin d'exécuter les applications Java), mais aussi un ensemble d'outils pour compiler et déboguer le code.

Nous avons utilisé JDK 1.6.

4.3.2 Eclipse, l'ADT et le SDK d'Android

L'adresse « http://developer.android.com/sdk/index.html » met à disposition un fichier téléchargeable gratuitement, que nous utilisons, et contenant un ensemble d'outils indispensables pour développer notre application Android. Ce paquet contient :

- ***Eclipse***, un environnement de développement spécialisé dans le développement Java mais qui n'est pas capable de développer des applications Android sans le composant suivant ;
- ***Le plugin ADT (Android Development Tools)***, qui est une extension d'Eclipse afin de développer des applications Android ;
- Des outils pour gérer l'installation d'Android sur le système.

Un IDE (Integrated Development Environment) est un logiciel dont l'objectif est de faciliter le développement, généralement pour un ensemble restreint de langages. En d'autres termes, il est possible de développer sans un IDE, mais en utiliser un est beaucoup plus pratique. En effet, il contient un certain nombre d'outils, dont au moins *un éditeur de texte* (souvent étendu pour avoir des fonctionnalités avancées telles que l'auto-complétion ou la génération automatique de code), *des outils de compilation* et *un débogueur*. Dans le cas du développement Android, un IDE est très pratique pour ne pas avoir à utiliser les lignes de commande. Nous utilisons ***Eclipse*** : en effet il est fourni par défaut par Google dans le paquetage que nous utilisons (il est gratuit, puissant et recommandé par Google dans la documentation officielle d'Android). Il est possible aussi d'opter pour d'autres IDE compétents tels que IntelliJ IDEA ou NetBeans [19].

En ce qui concerne le développement pour Android, le plug-in (l'extension) ***Android Development Tools*** (ou ADT) aidera à créer des projets pour Android avec les fichiers de base, mais aussi à tester, à déboguer et à exporter notre projet au format APK (pour pouvoir publier l'application).

Un SDK (Software Development Kit), c'est-à-dire un kit de développement, est un ensemble d'outils que met à disposition un éditeur afin de nous permettre de développer des applications pour un environnement précis. Le SDK Android permet donc de développer des applications pour Android et uniquement pour Android.

Figure 4.01 : *Splash screen d'Eclipse-ADT*

Figure 4.02 : *La barre d'outils d'Eclipse*

Figure 4.03 : *Icônes réservées à la SDK et à l'Android Virtual Device AVD*

Cliquer sur le bouton Android SDK Manager permet d'ouvrir l'outil de gestion du SDK d'Android.

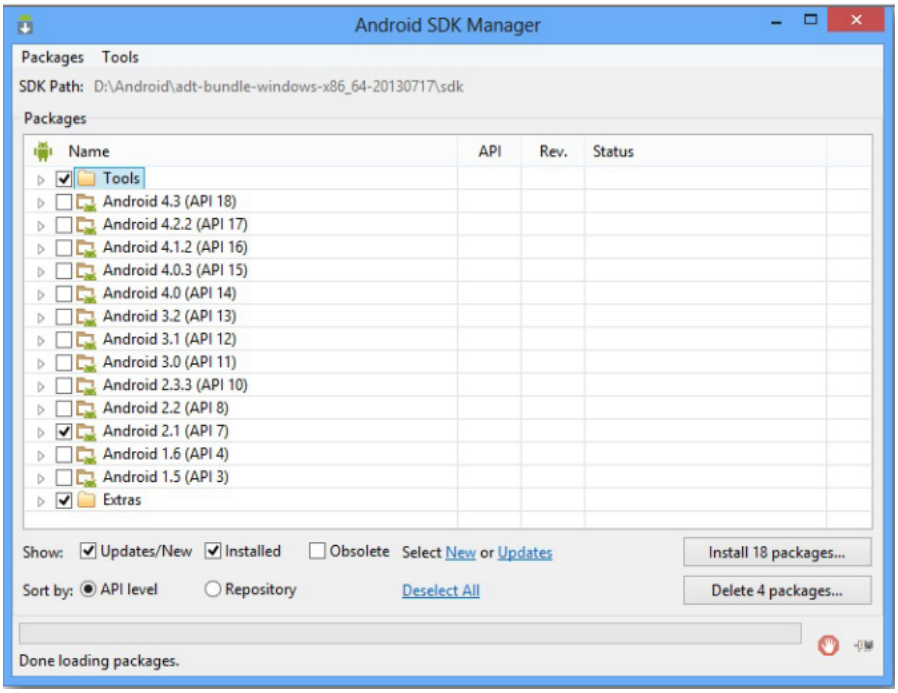


Figure 4.04 : *SDK Manager d'Android*

En ce qui concerne le nom des paquets, nous pouvons remarquer qu'ils suivent tous un même motif. Il est écrit à chaque fois Android [un nombre] (API [un autre nombre]). La présence de ces nombres s'explique par le fait qu'il existe plusieurs versions de la plateforme Android qui ont été développées depuis ses débuts et qu'il existe donc plusieurs versions différentes en circulation.

Le premier nombre correspond à la version d'Android et le second à la version de l'API Android associée. Quand on développe une application, il faut prendre en compte ces numéros, puisqu'une application développée pour une version précise d'Android ne fonctionnera pas pour les versions précédentes [19].

Nous avons choisi de délaisser les versions précédant la version 2.2 (l'API 8), de façon à ce que l'application puisse fonctionner pour 2.2, 3.1… mais pas forcément pour 1.6 ou 1.5.
Les API dont le numéro est compris entre 11 et 13 sont théoriquement destinées aux tablettes graphiques. En théorie, les applications développées avec les API numériquement inférieures fonctionneront, mais il y aura des petits efforts à fournir en revanche en ce qui concerne l'interface graphique.

Il est peut être injuste de laisser de côté les personnes qui sont contraintes d'utiliser encore ces anciennes versions, mais il faut savoir qu'ils ne représentent que 0,5 % du parc mondial des utilisateurs d'Android. De plus, les changements entre la version 1.6 et la version 2.2 sont trop importants pour être ignorés. Ainsi, l'application que nous développerons fonctionnera sous Android 2.2 minimum.

4.3.3 Configuration de l'émulateur de téléphone AVD et du terminal réel

L'Android Virtual Device, aussi appelé **AVD**, est un émulateur de terminal sous Android utile pour développer et tester l'application. Voir la figure 4.03 pour l'icône de lancement.

Figure 4.05 : *Terminal virtuel*

Rappelons cependant que l'émulateur ne propose pas toutes les fonctionnalités d'un vrai téléphone. Il ne permet par exemple pas d'émuler la gestion du Bluetooth. La machine est lourde à utiliser, voire très lourde sur les machines les plus modestes ; Il est beaucoup moins confortable à manipuler qu'un vrai terminal sous Android. Il faut aller dans la configuration des paramètres du téléphone virtuel, entrer dans l'option « Développement » puis cocher « Débogage USB » pour rendre le terminal apte à la programmation.

Configuration du terminal réel : Il faut configurer le téléphone de la même façon que l'émulateur. En plus, il faut indiquer que nous acceptons les applications qui ne proviennent pas du Market dans Configuration > Application > Source inconnue.
Pour les utilisateurs de Windows, dans notre cas, il faut d'abord télécharger les drivers adaptés à notre terminal.

4.4 Contenu d'un programme Android

Android utilise les mêmes concepts que la programmation classique, mais proposés de façon différente, avec une structure permettant de mieux protéger le fonctionnement des téléphones. Les composants principaux d'une application Android sont [22]:

Les activités (activities) : Ce sont les briques de base de l'interface utilisateur. Une activité peut être considérée comme l'équivalent Android de la fenêtre ou de la boîte de dialogue d'une application classique. Bien que des activités puissent ne pas avoir d'interface utilisateur, un code "invisible" sera délivré le plus souvent sous la forme de fournisseurs de contenus (content provider) ou de services.

Les fournisseurs de contenus (content providers) : Ils offrent un niveau d'abstraction pour toutes les données stockées sur le terminal, accessibles aux différentes applications. Le modèle de développement Android encourage la mise à disposition de ses propres données aux autres programmes ; construire un fournisseur de contenus permet d'obtenir ce résultat tout en gardant un contrôle total sur la façon dont on accédera aux données.

Les services : Les activités et les fournisseurs de contenus ont une durée de vie limitée et peuvent être éteints à tout moment. Les services sont en revanche conçus pour durer et, si nécessaire, indépendamment de toute activité. Nous pouvons, par exemple, utiliser un service pour vérifier les mises à jour d'un flux RSS (le terme RSS ou Really Simple Syndication signifie que le contenu du fichier est informatiquement codé selon le standard RSS, qui s'appuie lui-même sur le langage

informatique XML ou Extensible Markup Language) ou pour jouer de la musique, même si l'activité de contrôle n'est plus en cours d'exécution.

Les intentions (intents) : Ce sont des messages du système émis par le terminal pour prévenir les applications de la survenue de différents événements, que ce soit une modification matérielle (comme l'insertion d'une carte SD, SD pour Secure Digital) ou l'arrivée de données (telle la réception d'un SMS), en passant par les événements des applications elles-mêmes (notre activité a été lancée à partir du menu principal du terminal, par exemple). Nous pouvons non seulement répondre aux intentions, mais également créer les nôtres afin de lancer d'autres activités ou pour nous prévenir qu'une situation particulière a lieu.

4.5 Cycle de vie d'une activité

Pour développer d'une application sur Android, nous devons comprendre le cycle de vie d'une activité. Le cycle de vie d'une activité est exprimé par la figure suivant :

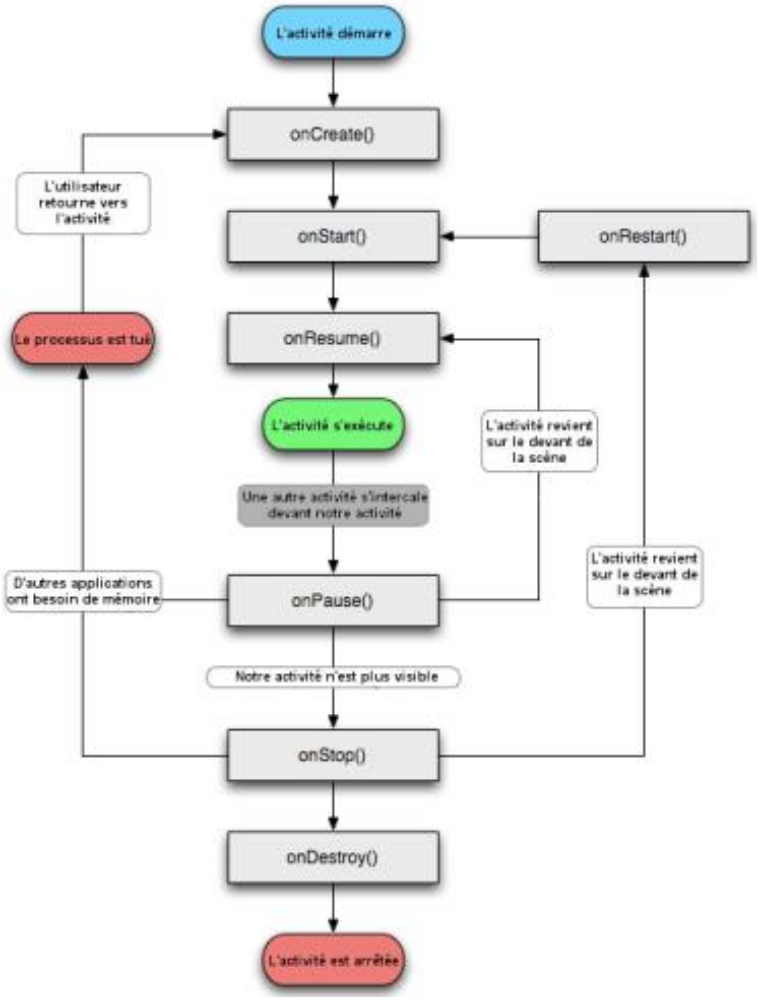


Figure 4.06 : *Cycle de vie d'une activité*

En général, une activité se trouve toujours dans l'un des quatre états suivants [22]:

- ***Active :*** L'activité a été lancée par l'utilisateur, elle s'exécute au premier plan. C'est à cet état que l'on pense quand on évoque le fonctionnement d'une activité.
- ***En pause :*** L'activité a été lancée par l'utilisateur, elle s'exécute et elle est visible, mais une notification ou un autre événement occupe une partie de l'écran. Pendant ce temps, l'utilisateur voit l'activité mais peut ne pas être capable d'interagir avec elle. Lorsqu'un appel téléphonique est reçu, l'utilisateur a l'opportunité de prendre cet appel ou de l'ignorer, par exemple.
- ***Stoppée :*** l'activité a été lancée par l'utilisateur, elle s'exécute mais est cachée par d'autres activités qui ont été lancées ou vers lesquelles le système a basculé. Notre application ne pourra rien présenter d'intéressant à l'utilisateur directement : elle ne peut passer que par une notification.
- ***Morte :*** l'activité n'a jamais été lancée (le téléphone vient d'être réinitialisé, par exemple) ou elle a été tuée, éventuellement à cause d'un manque de mémoire.

Il existe trois boucles principales :

La durée de vie d'une activité se passe entre le premier appel à ***OnCreate ()*** et l'appel à ***onDestroy ()***. Une activité met en place tous les états globaux dans la méthode ***onCreate ()*** et libère toutes les ressources restantes à ***onDestroy ()***.

La durée de vie visible d'une activité se passe entre un appel à ***onStart ()*** jusqu'à un appel correspondant à ***onStop ()***, durant laquelle l'utilisateur peut voir l'activité sur l'écran, même si elle n'est pas à l'avant et à l'interaction avec l'utilisateur. Entre ces deux méthodes, les ressources qui sont nécessaires pour montrer l'activité de l'utilisateur sont conservées.

La durée de vie d'une activité en avant-plan se passe entre un appel à ***onResume ()*** jusqu'à un appel correspondant à ***onPause ()***, durant laquelle l'activité est en face de toutes les autres activités afin d'interagir avec l'utilisateur. Une activité peut souvent changer son état entre l'état de reprise et l'état en pause.

4.6 Analyse et Conception

Idéalement, implémenter quelques techniques de stéganographie sur Android est l'objectif principal. L'application Android développée devra être capable de cacher un message secret ou une image secrète dans une image de couverture. L'application devra permettre de choisir deux

algorithmes de stéganographie, LSB (simple) et F5 (complexe) pour cacher le texte, préalablement chiffré AES, et un seul algorithme, le LSB, simple et rapide à implémenter pour cacher une image dans une image. L'encodage et le décodage doivent être faits en temps réel, c'est-à-dire en quelques secondes. Selon les actions de l'utilisateur, l'application doit l'avertir des erreurs et des champs incomplets, par d'éventuelles notifications. En outre, l'application devra être par défaut en langue anglaise, cependant, selon la configuration du terminal, permet de choisir la langue française.

4.6.1 Identification des besoins fonctionnels

L'application est construite à l'aide de groupe de modules. Chaque module effectue une certaine tâche. Les besoins basés sur chaque tâche sont appelées besoins fonctionnels.

En analysant les spécificités de l'application, et en gardant à l'esprit les besoins impératifs de l'utilisateur, l'idée est de développer une application simple, intuitive, design et exécutant les tâches le plus rapidement possible, et donc comportant six fonctions :

- Encoder un texte dans une image suivant la technique LSB
- Extraire un texte à partir d'un stégo-image encodé LSB.
- Encoder un texte dans une image suivant l'algorithme F5.
- Extraire un texte à partir d'un stégo-image encodé F5.
- Encoder une image dans une image suivant la technique LSB.
- Extraire une image à partir d'un stégo-image encodé LSB

Ces principales fonctions étant définies, voici d'autres fonctionnalités utiles de l'application :

- Cryptage du texte en AES à l'aide d'un mot de passe
- Partage de l'image traitée après opération.

La spécification et la validité des entrées seront expliquées plus loin.

4.6.2 Identification des besoins non fonctionnels

Les besoins non fonctionnels décrivent toutes les contraintes auxquelles est soumise l'application pour sa réalisation et son bon fonctionnement :

- ***Ergonomie et souplesse*** : l'application doit offrir une interface conviviale et ergonomique exploitable par l'utilisateur.

- ***Rapidité :*** l'application doit optimiser les traitements pour avoir un court temps de réponse.
- ***Efficacité et fiabilité :*** l'application doit remplir ses fonctions indépendamment de toutes circonstances pouvant entourer l'utilisateur de manière fiable, et ainsi garantir la confidentialité, l'imperceptibilité et l'intégrité de l'information cachée.
- ***Maintenabilité et évolutivité :*** le code de l'application doit être lisible et compréhensible afin d'assurer son état évolutif et extensible par rapport aux besoins du marché.

Pour décrire la conception de l'application, nous verrons le diagramme de cas d'utilisation et le diagramme d'activité. Par la suite nous passerons au diagramme de classe dans la section implémentation.

4.6.3 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est une représentation du comportement du système de point de vue de l'utilisateur, c'est une définition des besoins qu'attend un utilisateur du système, il contient tous les cas d'utilisation en liaison directe ou indirecte avec les acteurs. Il aide à gérer la complexité de l'application. Toutes les fonctionnalités requises sont décomposées en plusieurs petits scénarios.

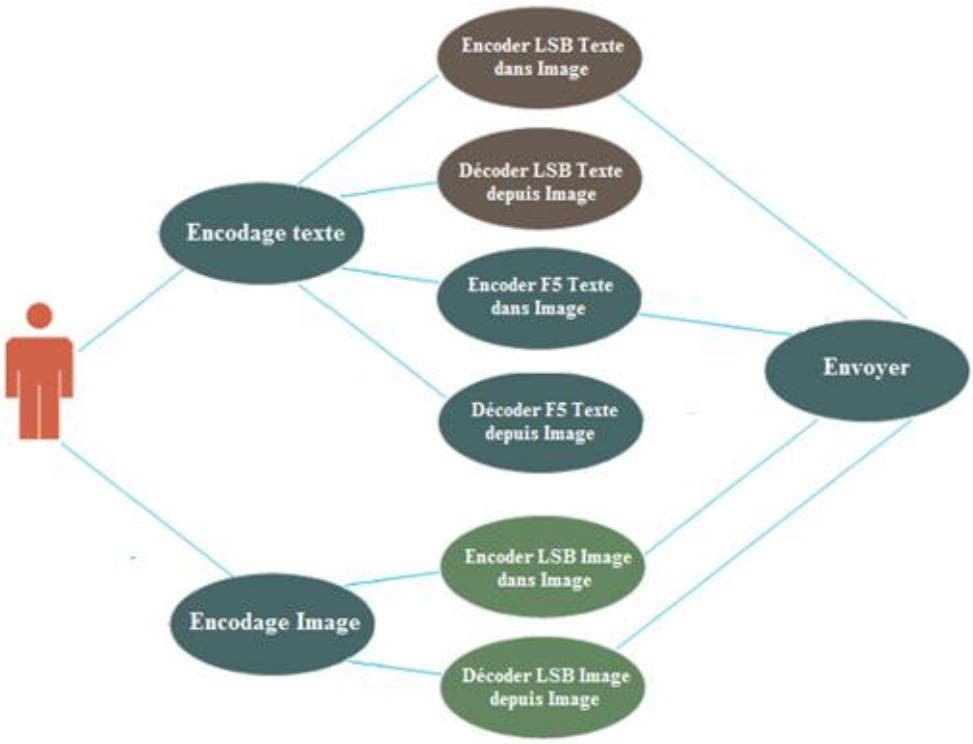


Figure 4.07 : *Diagramme de cas d'utilisation de l'application*

Cas d'utilisation 1	**Encodage Texte**
Intérêts et objectif	L'onglet « Encodage Texte » affiche les fonctionnalités disponibles pour la stéganographie de texte dans une image de l'application. En outre, l'utilisateur peut naviguer librement vers l'onglet « Encodage Image ».
Préconditions	
Post-conditions réussis	Une vue générale des options pour cette rubrique de l'application est présentée avec une navigation facile.
Post-conditions en échec	
Scénario nominal	Etape 1 : Après le lancement de l'application ou à partir de l'onglet « Encodage Image », l'utilisateur arrive à cet onglet. Etape 2 : L'utilisateur peut passer à l'étape suivante ou aller vers l'onglet « Encodage d'image ».

Tableau 4.01 : *Cas d'utilisation n°1*

Cas d'utilisation 2	**Encodage Image**
Intérêts et objectif	L'onglet « Encodage » Image affiche les fonctionnalités disponibles pour la stéganographie d'image dans une image de l'application. En outre, l'utilisateur peut naviguer librement vers l'onglet « Encodage Texte ».
Préconditions	
Post-conditions réussis	Une vue générale des options pour cette rubrique de l'application est présentée avec une navigation facile.
Post-conditions en échec	
Scénario nominal	Etape 1 : Après le lancement de l'application, l'utilisateur choisit l'onglet « Encodage Image ». Etape 2 : L'utilisateur peut passer à l'étape suivante ou aller vers l'onglet « Encodage Texte ».

Tableau 4.02 : *Cas d'utilisation n°2*

Cas d'utilisation 3	**Encoder LSB Texte dans Image**
Intérêts et objectif	Cacher un message secret dans une image avec la technique LSB.
Préconditions	
Post-conditions réussis	Génération d'un stégo-image au format png, contenant le message
Post-conditions en échec	Message d'erreur pour une image invalide ou champs incomplets
Scénario nominal	Etape 1 : L'utilisateur démarre l'application, Etape 2 : entre dans l'onglet « Encodage Texte », Etape 3 : choisit l'image de couverture, soit dans la galerie d'images, soit en cliquant sur le bouton « Appareil » pour pendre une photo à l'aide de l'appareil photo du terminal Etape 4 : entre le message secret, Etape 5 : entre un mot de passe si nécessaire, Etape 6 : choisit l'algorithme LSB, Etape 7 : choisit l'opération « Encoder ». Etape 8 : clique le bouton « Lancer ».

Tableau 4.03 : *Cas d'utilisation n°3*

Cas d'utilisation 4	**Décoder LSB Texte depuis Image**
Intérêts et objectif	Extraire un message secret dans un stégo-image LSB.
Préconditions	
Post-conditions réussis	Affichage du message secret
Post-conditions en échec	Renvoi d'un message d'erreur pour une image ou un mot de passe invalide, ou absence de message dans l'image
Scénario nominal	Etape 1 : L'utilisateur démarre l'application, Etape 2 : entre dans l'onglet « Encodage Texte », Etape 3 : choisit le stégo-image dans la galerie d'images, Etape 4 : entre un mot de passe si nécessaire, Etape 5 : choisit l'algorithme LSB, Etape 6 : choisit l'opération « Décoder », Etape 7 : clique le bouton « Lancer ».

Tableau 4.04 : *Cas d'utilisation n°4*

Cas d'utilisation 5	**Encoder F5 Texte dans Image**
Intérêts et objectif	Cacher un message secret dans une image avec l'algorithme F5.
Préconditions	
Post-conditions réussis	Génération d'un stégo-image au format jpg, contenant le message
Post-conditions en échec	Renvoi d'un message d'erreur pour une image invalide ou des champs incomplets
Scénario nominal	Etape 1 : L'utilisateur démarre l'application, Etape 2 : entre dans l'onglet « Encodage Texte », Etape 3 : choisit l'image de couverture, soit dans la galerie d'images, soit en prenant une photo à l'aide de l'appareil photo, Etape 4 : entre le message secret, Etape 5 : entre un mot de passe si nécessaire, Etape 6 : choisit l'algorithme F5, Etape 7 : choisit l'opération « Encoder ». Etape 8 : clique le bouton « Lancer ».

Tableau 4.05 : *Cas d'utilisation n°5*

Cas d'utilisation 6	**Décoder F5 Texte depuis Image**
Intérêts et objectif	Extraire un message secret dans un stégo-image F5.
Préconditions	
Post-conditions réussis	Affichage du message secret
Post-conditions en échec	Renvoi d'un message d'erreur pour une image ou un mot de passe invalide, ou absence de message dans l'image
Scénario nominal	Etape 1 : L'utilisateur démarre l'application, Etape 2 : entre dans l'onglet « Encodage Texte », Etape 3 : choisit le stégo-image dans la galerie d'images, Etape 4 : entre un mot de passe si nécessaire, Etape 5 : choisit l'algorithme F5, Etape 6 : choisit l'opération « Décoder », Etape 7 : clique le bouton « Lancer ».

Tableau 4.06 : *Cas d'utilisation n°6*

Cas d'utilisation 7	**Encoder LSB Image dans Image**
Intérêts et objectif	Cacher une image dans une image avec l'algorithme LSB.
Préconditions	
Post-conditions réussis	Génération d'un stégo-image au format png, contenant l'image secrète
Post-conditions en échec	Renvoi d'un message d'erreur pour des images invalides
Scénario nominal	Etape 1 : L'utilisateur démarre l'application, Etape 2 : entre dans l'onglet « Encodage Image », Etape 3 : choisit l'image de couverture dans la galerie d'images en cliquant sur le bouton correspondant, Etape 4 : choisit l'image secrète à cacher dans la galerie d'images en cliquant sur le bouton correspondant, Etape 5 : choisit l'opération « Encoder », Etape 6 : clique le bouton « Lancer ».

Tableau 4.07 : *Cas d'utilisation n°7*

Cas d'utilisation 8	**Décoder LSB Image depuis Image**
Intérêts et objectif	Extraire une image secrète dans un stégo-image LSB.
Préconditions	
Post-conditions réussis	Génération et affichage de l'image secrète cachée au format png.
Post-conditions en échec	Renvoi d'un message d'erreur pour l'image invalide, ou absence d'image secrète dans l'image choisie
Scénario nominal	Etape 1 : L'utilisateur démarre l'application, Etape 2 : entre dans l'onglet « Encodage Image », Etape 3 : choisit le stégo-image dans la galerie d'images, Etape 4 : choisit l'opération « Décoder », Etape 5 : clique le bouton « Lancer ». Etape 6 : choisit l'opération « Décoder », Etape 7 : clique le bouton « Lancer ».

Tableau 4.08 : *Cas d'utilisation n°8*

Cas d'utilisation 9	**Envoyer**
Intérêts et objectif	Partager le stégo-image généré ou une image extraite.
Préconditions	Au moins une application de partage, par exemple via WiFi, Bluetooth, email, MMS ou autres, est installée sur le terminal.
Post-conditions réussis	L'image partagée est effectivement reçue par un autre terminal ou prise en charge par une application choisie par l'utilisateur.
Post-conditions en échec	Aucune application permettant de prendre en charge l'image à partager n'est installée sur le terminal
Scénario nominal	Etape 1 : L'utilisateur démarre l'application, Etape 2 : entre dans l'onglet « Encodage Image » ou « l'onglet Encodage Texte », Etape 3 : effectue une opération, Etape 4 : clique sur le bouton « Envoyer », Etape 5 : choisit une application proposée par l'application

Tableau 4.09 : *Cas d'utilisation n°9*

4.6.4 Diagramme d'activité

Le diagramme d'activité doit représenter l'ensemble des actions réalisées par le système, avec tous les branchements conditionnels et toutes les boucles possibles. C'est un graphe orienté d'actions et de transitions. Les transitions sont franchies lors de la fin des actions ; des étapes peuvent être réalisées en parallèle ou en séquence.

L'application commence par l'utilisateur qui choisit l'encodage de texte ou d'image. Ensuite, si l'utilisateur, après avoir choisi l'encodage texte désire encoder un message secret, choisit d'abord une image de couverture, un éventuel mot de passe, un algorithme de stéganographie, F5 ou LSB, l'option encodage, puis lance le traitement.

Le stégo-image généré peut être ensuite partagé par Wi-Fi (Wireless Fidelity), Bluetooth ou pris en charge par d'autres applications installées sur le terminal. L'utilisation des autres fonctionnalités est assez intuitive. Il faut seulement remarquer que pour l'encodage d'image dans une image, seul l'algorithme LSB est implémenté en raison de sa simplicité et de la rapidité de la technique.

Des entrées invalides ou manquantes font appel à un message d'erreur.

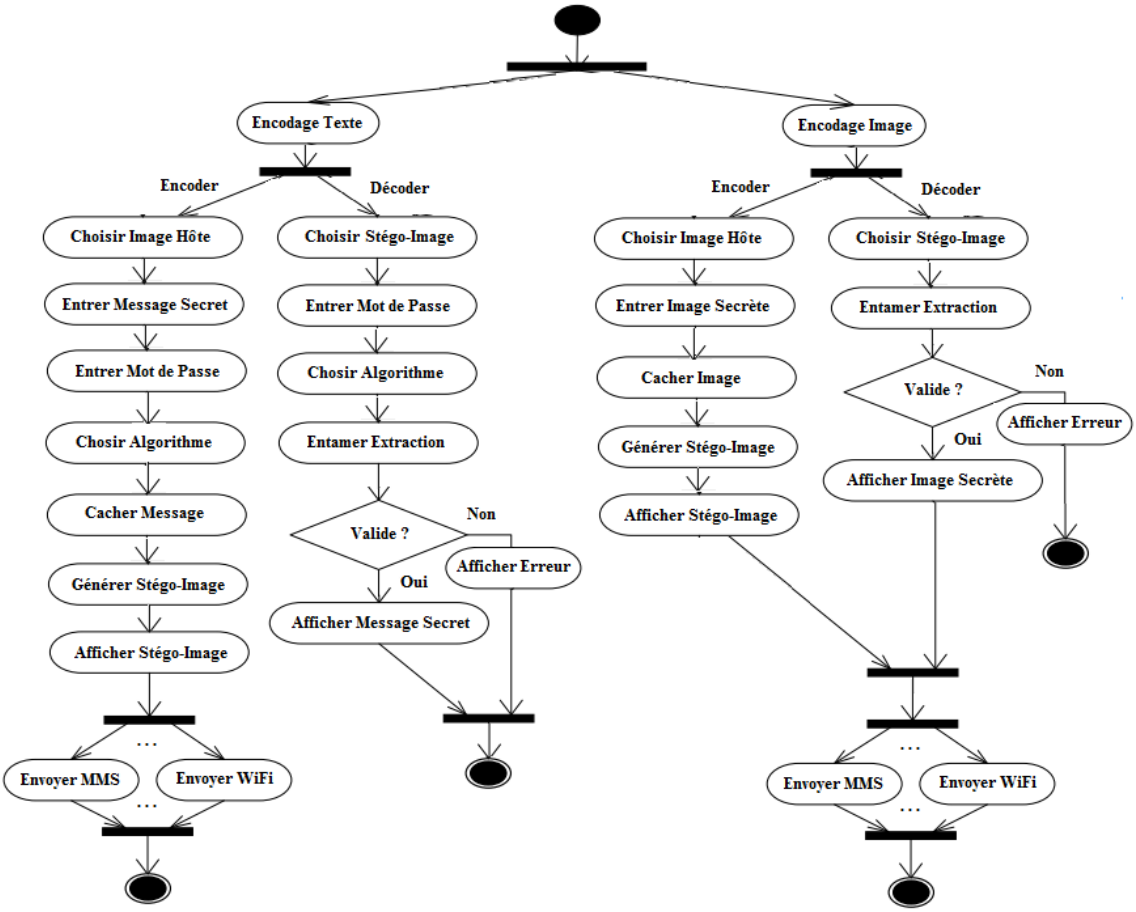


Figure 4.08 : *Diagramme d'activité de l'application*

4.6.5 Techniques utilisées

L'application aura donc comme principales fonctions l'encodage de texte préalablement chiffré dans une image, le décodage de texte avec déchiffrement à partir d'un stégo-image, l'encodage d'image dans une image, et l'extraction d'image secrète à partir d'un stégo-image.

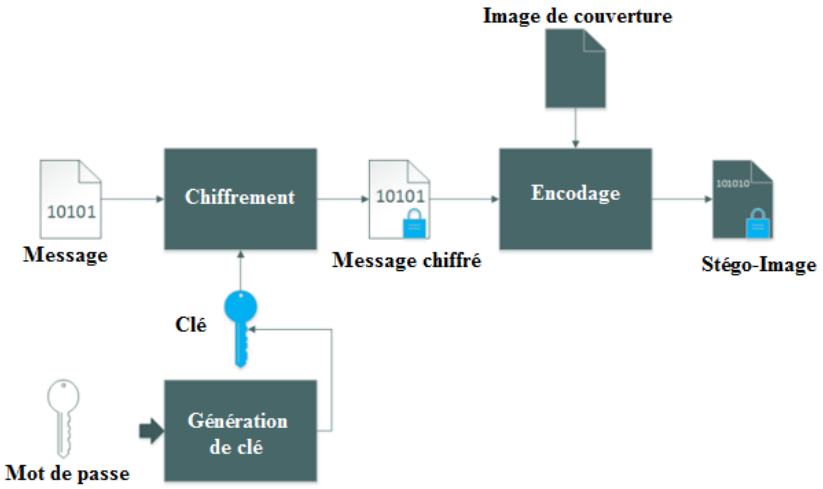


Figure 4.09 : *Processus stéganographique d'encodage de texte préalablement chiffré*

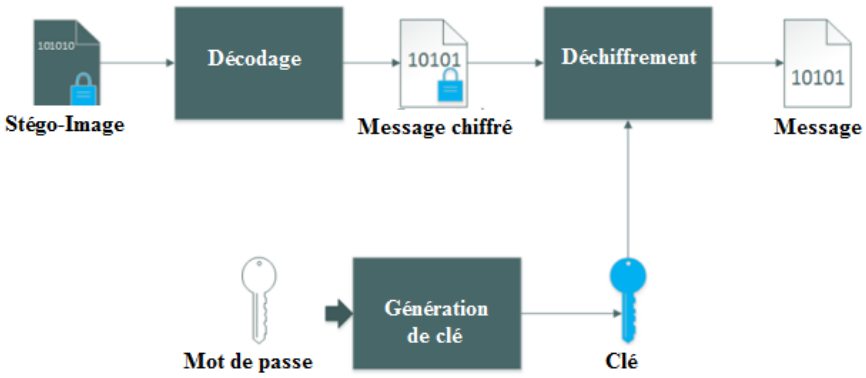


Figure 4.10 : *Processus stéganographique de décodage de texte à partir d'un stégo-image avec déchiffrement*

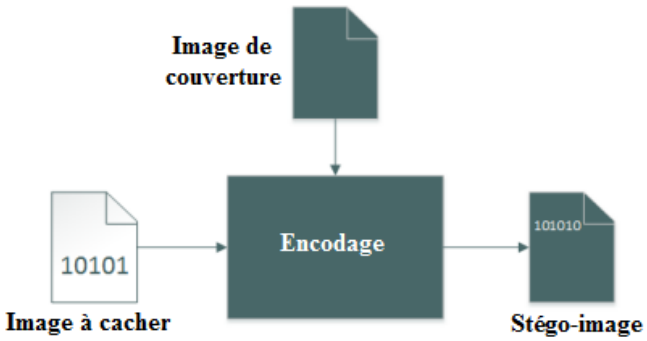


Figure 4.11 : *Processus stéganographique d'encodage d'image dans une image*

Figure 4.12 : *Processus stéganographique d'extraction d'image à partir d'un stégo-image*

Le texte sera crypté en AES, et les algorithmes de stéganographie utilisés : le LSB ou F5 pour l'encodage de texte, et LSB pour l'encodage d'image.

Chiffrement AES : La méthode de chiffrement utilisée pour le texte sera AES dont la longueur de la clé est de 128 bits. Cette clé sera générée à partir du mot de passe fourni par l'utilisateur.

Technique LSB : cette technique déjà présentée auparavant consiste à modifier les derniers bits du codage des couleurs de l'image de couverture pour y insérer le texte chiffré (ou les informations

de l'image à cacher pour l'encodage d'image dans une image). Les figures suivantes montrent la structure du paquet encodé pour l'encodage de texte et l'encodage d'image :

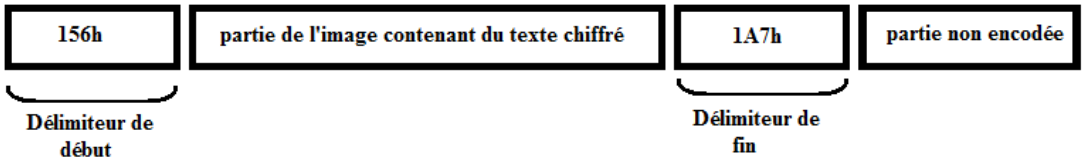


Figure 4.13 : *Format du paquet encodé LSB par un texte secret crypté AES*

Figure 4.14 : *Format du paquet encodé LSB par une image secrète*

Il y a quelques points essentiels à savoir :

- Nous insérons des délimiteurs de début (156h) et de fin (1A7h) pour faire savoir à l'application qu'un texte (ou une image) est caché dans l'image choisie par l'utilisateur, et plus précisément dans quelle partie de l'image, ce qui sera utile pour le décodage.
- Pour l'encodage de texte, le message sera toujours chiffré pour assurer une certaine confidentialité, même si l'utilisateur n'entre pas de mot de passe (un mot de passe par défaut sera utilisé pour générer la clé).
- Pour l'encodage d'image, la largeur et la hauteur de l'image à cacher seront encodées en premier, suivi de ses données proprement dites, dans l'image de couverture.

Algorithme F5 : cette technique un peu plus complexe a aussi été déjà détaillée auparavant. Elle utilise la compression JPEG, elle cache le message (préalablement chiffré) à travers les coefficients DCT de l'image de couverture. Le paramètre p et la longueur du texte chiffré seront encodée en premier par une technique LSB, puis le message chiffré proprement dit sera encodé par la technique de matrix embedding (cf. annexe 1). Voici la structure du paquet encodé F5 :

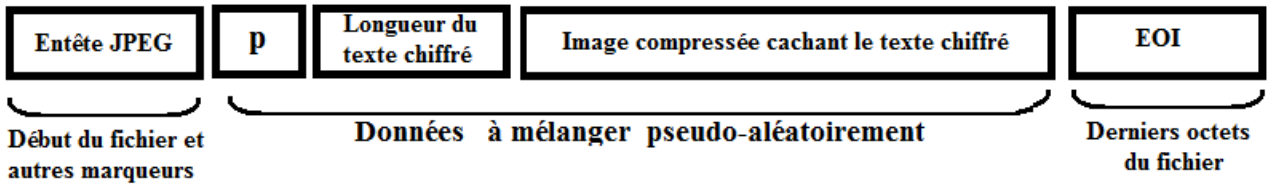


Figure 4.15 : *Format du paquet encodé F5*

4.7 Implémentation

4.7.1 Spécifications du projet

Le fichier "AndroidManifest.xml" (cf. annexe 4) contient toutes les spécifications sur l'application Android y compris la structure du projet.

Nom de l'application	**MySteg**
SDK min	8
permissions	Ecriture sur la carte SD, appareil photo
Langues	Anglais par défaut, français disponible
Taille occupée sur le terminal	1.5 Mo
Autres	L'application fonctionne parfaitement hors ligne, mais pour le partage, il faut que l'utilisateur possède d'autres applications prenant en charge le fichier à partager par exemple des applications de transfert par bluetooth, WiFi, MMS, ….

Tableau 4.10 : *Spécifications de l'application MySteg*

4.7.2 Paramètres d'entrées et de sortie de l'application

Voici les paramètres d'entrées et de sortie valide :

	LSB	**F5**
Texte à encoder	Non vide, alphanumérique, et/ou caractères spéciaux	Non vide, alphanumérique, et/ou caractères spéciaux
Image secrète	Fichiers .png et .bmp	
Mot de passe	Vide, ou alphanumérique, et/ou caractères spéciaux	Vide, ou alphanumérique, et/ou caractères spéciaux
Image de couverture	Fichiers .png et .bmp recommandés, fichiers .jpg plus ou moins acceptables	Fichiers .png, .bmp, .jpg
Stégo-image généré	Fichiers .png	Fichiers .jpg

Tableau 4.11 : *Paramètres d'entrées et de sortie*

4.7.3 Diagramme de classe de l'application

Voici le diagramme de classe de l'application :

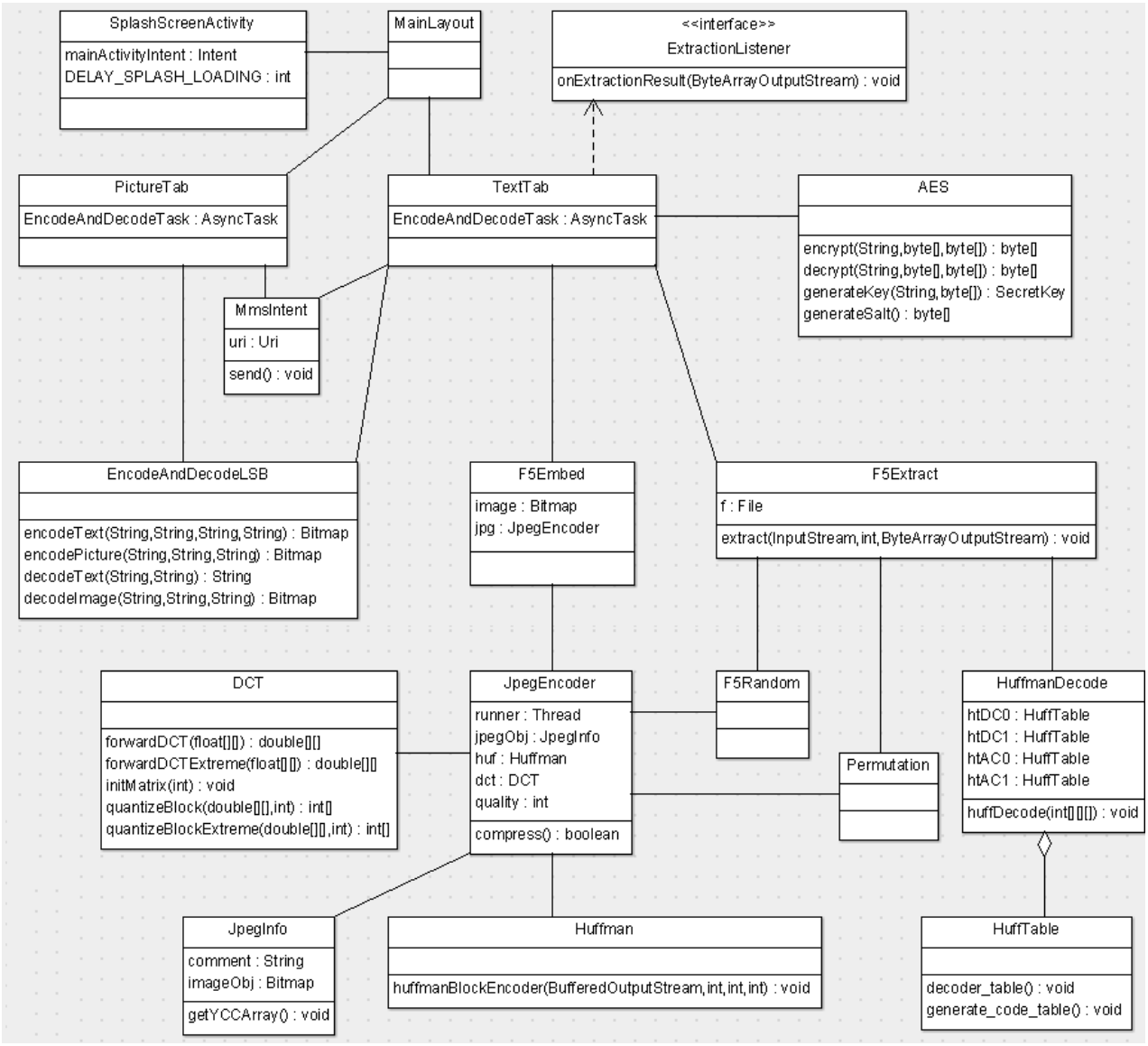


Figure 4.16 : *Diagramme de classe de l'application*

L'interface de l'application est assez simple, une classe héritant de la classe « Activity » applelée « SplashScreeActivity.java » fait apparaître un écran de démarrage, qui dure 2 secondes, amène à une activité « MainLayout.java » qui intègre deux onglets : « TextTab.java » et « PictureTab.java », deux onglets pour l'encodage de texte et l'encodage d'image. Notons que le design de l'interface graphique se fait à l'aide de fichiers xml situés dans le dossier « res/layout » du projet. Cette indépendance du code principal et de l'interface graphique facilite considérablement la modification future en vue d'une amélioration de l'application.

L'implémentation de l'algorithme LSB se fait par l'intermédiaire d'une classe « EncodeAndDecodeLSB.java », le cryptage AES par la classe « AES.java ».

Toutefois, l'implémentation de l'algorithme F5 est un peu plus complexe, et nécessite de le décomposer en plusieurs autres classes, par exemple pour l'encodage (F5Embed.java) qui utilise d'autres classes comme « JpegEncoder.java » utilisant d'autres classes comme « DCT.java ».

4.7.4 Interfaces graphiques de l'application

Au démarrage de l'application, nous pouvons choisir entre les deux onglets « Encodage Texte » et « Encodage d'image ».

Figure 4.17 : *Interfaces de démarrage de l'application MySteg*

Pour l'encodage de texte par exemple, voici quelques captures d'écran :

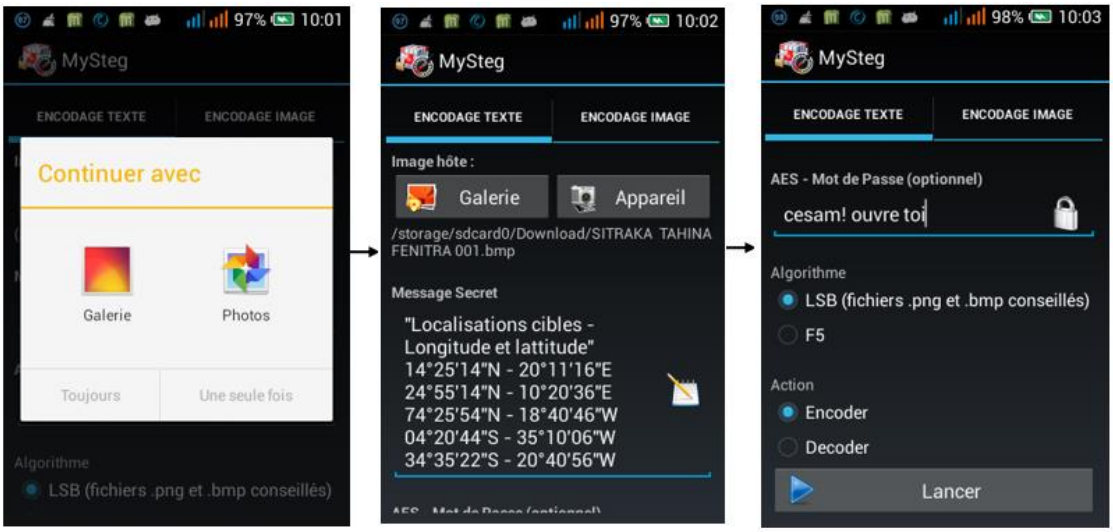


Figure 4.18 : *Captures d'écran pour l'encodage de texte dans une image*

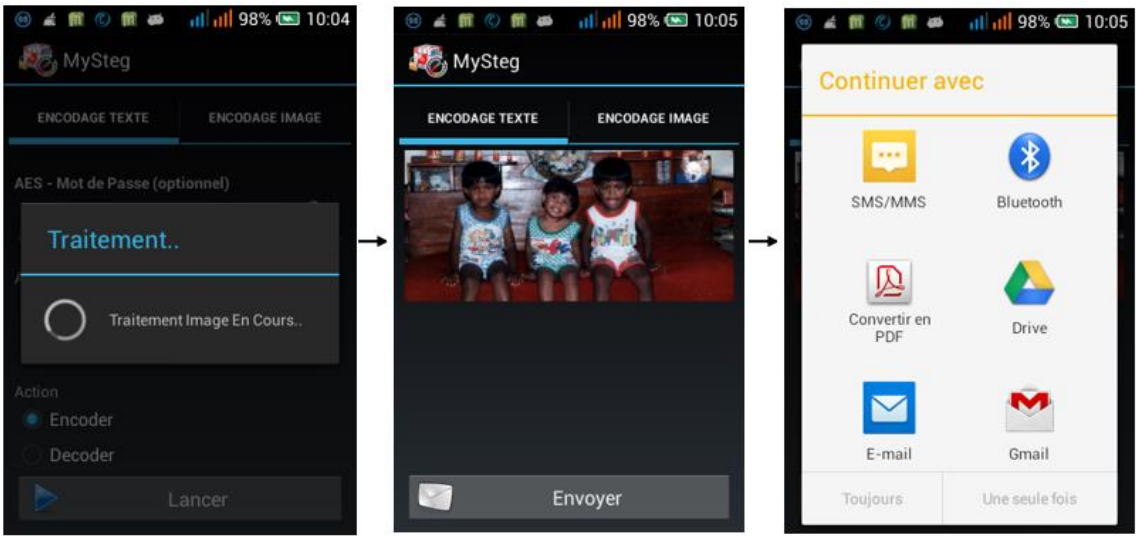


Figure 4.19 : *Génération du stégo-image et partage*

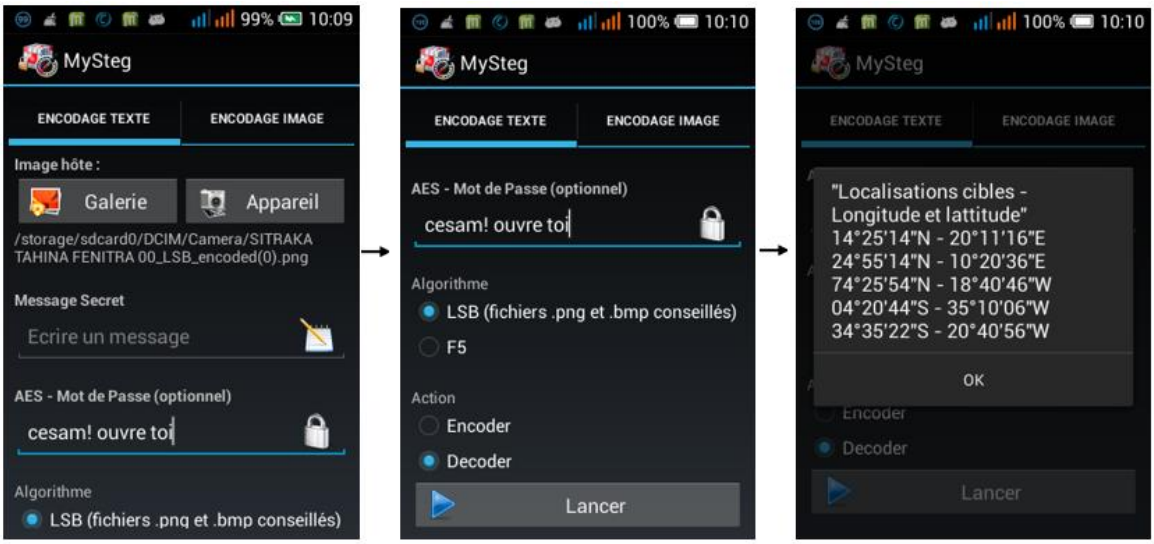


Figure 4.20 : *Extraction du message secret à partir du stégo-image*

4.8 Conclusion

Ce chapitre comprend les phases de conception et de développement de l'application. Une notion sur la programmation Android a été présentée, puis l'analyse des besoins, la conception et l'implémentation de l'application. L'application développée, qui a comme nom évocateur « MySteg », est donc une application de stéganographie, qui nous permet de cacher du texte préalablement chiffré AES, ou une image, dans une image de couverture, à l'aide de deux algorithmes LSB et F5. L'évaluation de la performance de l'application et les tests se feront dans le chapitre suivant.

CHAPITRE 5
EVALUATIONS ET TESTS DE L'APPLICATION

5.1 Introduction

Ce chapitre, également très important, concerne l'évaluation de la performance de l'application. Plusieurs tests ont été effectués à l'aide d'un programme que nous avons conçus sur MATLAB. Les tests effectués ont pour but de cerner les différentes subtilités de l'application, ainsi que sa performance. La première partie de ce chapitre décrit le programme d'évaluation, la deuxième partie concerne les tests proprement dits, et la dernière effectue un petit bilan sur les deux algorithmes de l'application, la technique LSB et l'algorithme F5.

5.2 Evaluation et test

Pour évaluer la performance de l'application, nous avons conçu un programme sous MATLAB version « R2013a (8.1.0.604) 64bit », acronyme de « MATrix LABoratory » (logiciel développé par Mathworks conçu pour les calculs scientifiques, possédant toutes les fonctionnalités des approches récentes de programmation).

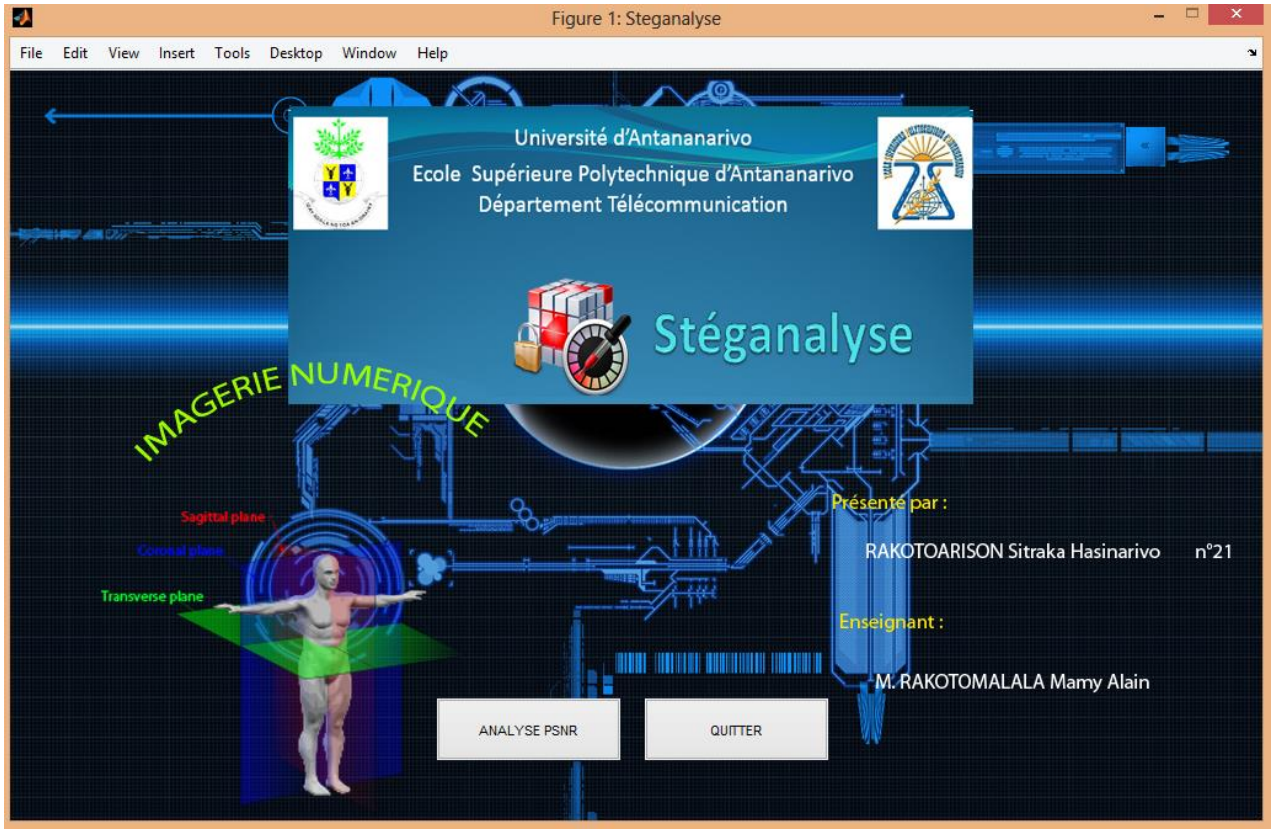


Figure 5.01 : *Interface principale du programme*

Après le lancement du programme, nous obtenons la fenêtre principale sur la figure 5.01, qui possède deux boutons : « ANALYSE PSNR » et « QUITTER ». L'appui sur le premier bouton amène à une deuxième fenêtre (cf. figure 5.02), dans laquelle nous pourrons faire une analyse PSNR (Peak Signal-to-Noise Ratio), une comparaison entre deux images, une image originale et une image traitée (les images étant impérativement de même dimensions). Le second bouton permettra de quitter le programme.

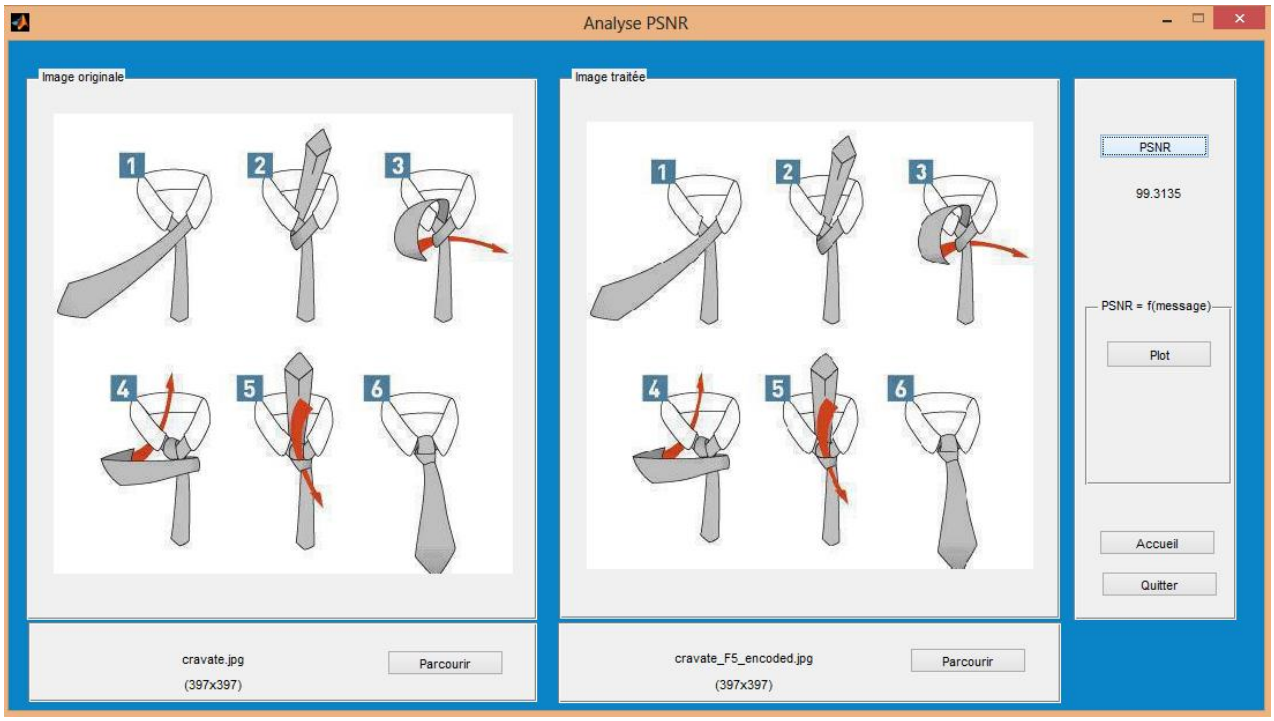


Figure 5.02 : *Fenêtre « Analyse PSNR » du programme*

La fenêtre « Analyse PSNR » du programme permet donc de charger deux images à l'aide des boutons « Parcourir », généralement l'image originale et le stégo-image, ou l'image secrète originale et l'image extraite à partir d'un stégo-image, afin de pouvoir calculer le PSNR des deux images à l'aide du bouton « PSNR », une fois ces images chargées. Sur la figure 5.02, le PSNR de l'image originale « cravate.jpg » et le stégo-image « cravate_F5_encoded.jpg » est de 99.31 dB. La manipulation de l'interface graphique est assez facile à comprendre. Le bouton « Plot », de la section « PSNR = f(message) » trace une courbe du PSNR en fonction de la taille du message caché dans le stégo-image. Nous verrons bientôt en détails la signification du PSNR, et comment le calculer pour deux images en RGB.

Le chargement d'une image sera effectué à l'aide des boutons « Parcourir » de chaque section, ce qui appellera une boîte de dialogue (cf. figure 5.03) permettant de choisir une image au format jpg, png ou bmp.

Le nom de l'image choisie, ainsi que ses dimensions s'afficheront dans la section appropriée, une fois le chargement effectué.

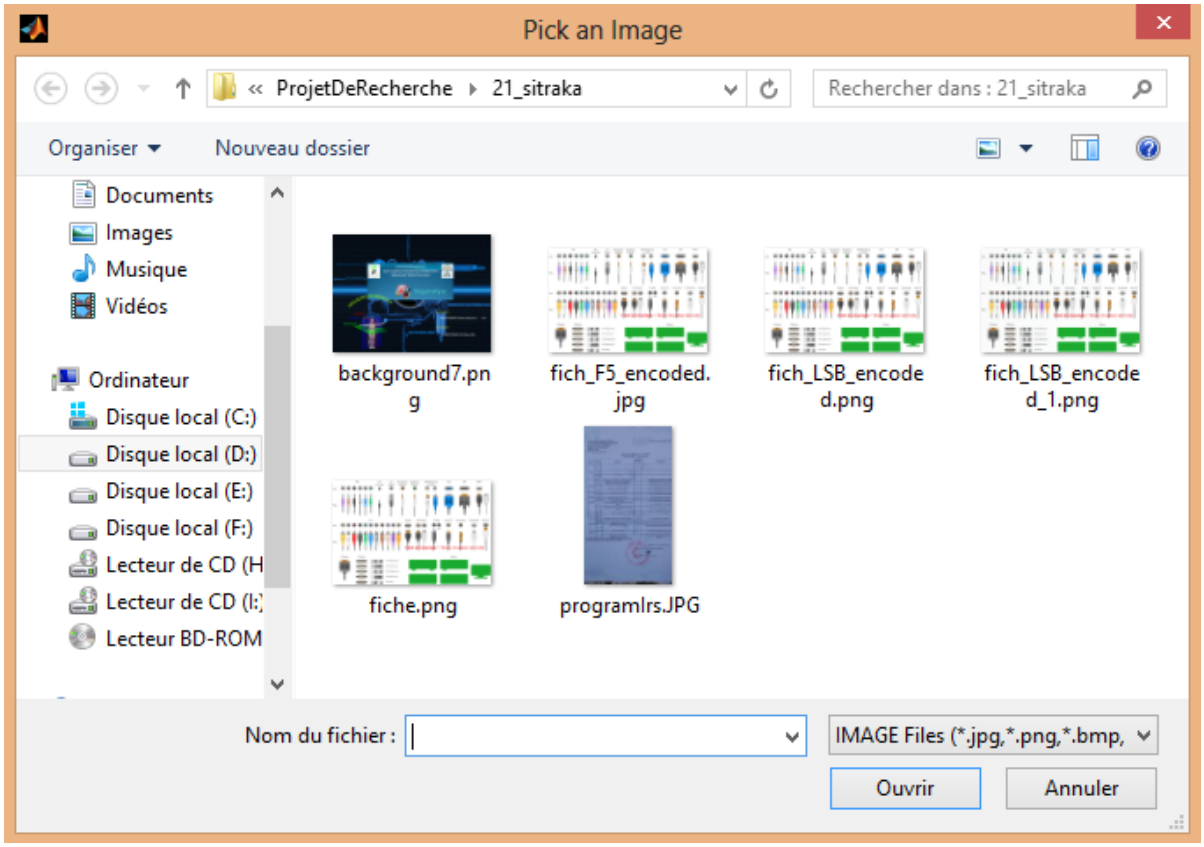


Figure 5.03 : *Boîte de dialogue pour le chargement d'une image*

5.2.1 Le PSNR

Le PSNR (Peak Signal-to-Noise Ratio) est un outil utilisé en imagerie numérique pour mesurer la distorsion entre une image originale, et l'image traitée. Son unité est le décibel (dB).

Pour deux images en niveaux de gris X et Y de même taille m x n, le PSNR est évalué comme suit [23]:

1. Nous calculons l'erreur quadratique moyenne MSE (Mean Square Error) entre les deux images :

$$MSE(X,Y) = \frac{1}{m \times n} \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} \left[X(i,j) - Y(i,j) \right]^2 \qquad (5.01)$$

2. Le PSNR est donné par :

$$\mathbf{PSNR(X,Y) = 10 \times \log_{10}\left(\frac{max(X)^2}{MSE(X,Y)}\right)} \quad (5.02)$$

Avec max(X) la valeur maximum possible pour le niveau de gris. Dans le cas standard d'une image où le niveau de gris est codé sur 8 bits, max(X) = 255.

Nous sommes intéressés par l'utilisation du PSNR pour comparer les images codées en RGB. Dans cette optique, nous considérons 2 images RGB : X(n, m, p) l'image originale et Y(n, m, p) le stégo-image par exemple (avec p = 3). La MSE est donnée par :

$$\mathbf{MSE(X,Y) = \frac{1}{m \times n \times p} \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} \sum_{k=1}^{k=p} \left[X(i,j,k) - Y(i,j,k)\right]^2} \quad (5.03)$$

Nous calculons la « dynamique » des deux images :

$$\mathbf{m_1 = argmax(X(n, m, p))} \quad (5.04)$$

$$\mathbf{m_2 = argmax(Y(n, m, p))} \quad (5.05)$$

Ensuite, nous choisissons la plus grande entre les deux valeurs :

$$\mathbf{D = max([m_1\ m_2])} \quad (5.06)$$

D'autres versions prennent 1 ou 255, au lieu de max(max(X), max(Y)).
Finalement, le PSNR est donné par :

$$\mathbf{PSNR(X,Y) = 10 \times \log_{10}\left(\frac{D^2}{MSE(X,Y)}\right)} \quad (5.07)$$

Plus le PSNR augmente, plus les deux images se ressemblent. Si le PSNR tend vers l'infini, les deux images sont carrément identiques. La valeur minimale du PSNR est de 30 dB, en dessous, le stégo-image est considéré comme trop dégradé, ce qui aura un impact visuel notable entre les deux images et éveiller les soupçons.

5.2.2 Tests, résultats et interprétations

Plusieurs tests ont été effectués, notamment sur plusieurs images différentes. Les caractéristiques du terminal avec lequel nous avons effectué les tests peuvent être lues dans le tableau 5.01.

Spécifications	One Touch 4015D (Alcatel Pixy 2)
Version d'Android	4.2.2 Jelly Bean
Resolution de l'écran (tactile) en pixels	320 x 480
Type de processeur	Dual Core 1 GHz
Mémoire vive	512 Mo
Mémoire interne	2 Go
Appareil photo	2 Mpx

Tableau 5.01 : *Caractéristiques techniques du terminal test*

De plus, il est utile ici de préciser le genre d'information ou le type de message qu'un utilisateur est susceptible de vouloir cacher. Voici donc des exemples de textes à cacher :

Message 1 : "Localisations cibles - Longitude et latitude"

14°25'14"N - 20°11'16"E
24°55'14"N - 10°20'36"E
74°25'54"N - 18°40'46"W
04°20'44"S - 35°10'06"W
34°35'22"S - 20°40'56"W

Message 2 : "Mes comptes et mot de passe :

RSitrakaH@yahoo.fr : RSitrakaYahooMDP0555
RSitrakaH@gmail.com : RSitrakaGMailMDP0555
Facebook : RSitrakaFacebookMDP0555

BaseDeDonneeEntreprise : AdminMDP0254PPnjl

OrdiPortablePersoAdmin : RSitrakaOrdiPersoMDP0555
OrdiPortableBureauAdmin : RSitrakaOrdiBureauMDP0555

Valise : 948
carte ID : 1550245268425586865

Message 3 : "Il y a un coffre-fort caché dans la maison derrière le tableau Picasso, dans la chambre. Code = 0215489212321"

Message 4 : Seules trois images parmi les dix que je t'ai envoyés avec cette image sont aussi des stégo-images : fiche.png, ours.png et vacance.jpg

- fiche.png : encodage texte, algorithme LSB, mot de passe : instructionMDP
- vacance.jpg : encodage texte, algorithme F5, mot de passe : suiteInstructionMDP
- ours.png : encodage image, algorithme LSB, mot de passe : imagePlanMDP

Message 5 : "Lutte pour la liberté - programme :

08/04/15 - 7h10 : Rassemblement général

Lieu : Ivandry Lot II J 554 Ter

Mot de passe : Luttons pour la nation"

Message 6 : "Ceci est un appel à l'aide....Nous sommes des touristes retenus par des terroristes, en Afghanistan, 15 Km vers le Nord de Kaboul, Nous étions 10 au départ, 6 hommes, 4 femmes...3 hommes sont morts... 1 homme blessé à la jambe...

Terroristes : 20 hommes armés

Veuillez nous secourir de toute urgence

Nous pouvons donc voir une certaine utilité de la stéganographie, que ce soit pour son compte personnel par exemple en cachant ses données importantes dans une image innocente, ou pour pouvoir s'échanger des messages ou des images confidentielles à l'abri des regards indiscrets. Mais l'utilisation de la stéganographie n'a de limites que celles de l'imagination de l'utilisateur.

En outre, il faut bien aussi choisir l'image de couverture pour ne pas éveiller les soupçons. De manière générale, il faudrait choisir des images innocentes qui n'attirent pas trop l'attention (par exemple des images de la nature, de paysages, des images normales du quotidien…), ou au moins qui la détourne, par exemple des images comiques, images de vacances, de fêtes…. Quelques images types sont proposées (cf. figure 5.04).

Figure 5.04 : *Exemples d'mages à choisir comme couverture*

Test 1 : Nous testons d'abord l'encodage texte. Pour cela, nous allons choisir différentes images de couverture, de tailles et de formats différents. Voici l'information à cacher et le résultat :

"Mes comptes et mots de passe" :
RSitrakaH@yahoo.fr : RSitrakaYahooMDP0555
RSitrakaH@gmail.com : RSitrakaGMailMDP0555
Facebook : RSitrakaFacebookMDP0555

BaseDeDonneeEntreprise : AdminMDP0254PPnjl

OrdiPortablePersoAdmin : RSitrakaOrdiPersoMDP0555
OrdiPortableBureauAdmin : RSitrakaOrdiBureauMDP0555

Valise : 948
Carte ID : 1550245268425586865

Images de couverture	**Stégo-images**	
	LSB	**F5**
Fiche.png (720 x 476) 128 Ko	Fiche_LSB_encoded.png (720 x 476) 134 Ko **PSNR** = 142.75 dB	Fiche_F5_encoded.jpg (720 x 476) 75.5 Ko **PSNR** = 78.78 dB
Bean.png (400 x 545) 447 Ko	Bean_LSB_encoded.png (400 x 545) 295 Ko **PSNR** = 150.38 dB	Bean_F5_encoded.jpg (400 x 545) 50.4 Ko **PSNR** = 97.32 dB
cravate.jpg (397 x 397) 17.2 Ko	cravate_LSB_encoded.png (397 x 397) 90.7 Ko **PSNR** = 102.12 dB	cravate_F5_encoded.jpg (397 x 397) 24.6 Ko **PSNR** = 99.31 dB
anniv.bmp (734 x 510) 1.07 Mo	anniv_LSB_encoded.png (734 x 510) 627 Ko **PSNR** = 154.6 dB	anniv_F5_encoded.jpg (734 x 510) 105 Ko **PSNR** = 80.49 dB

Tableau 5.02 : *Résultat du test 1*

Nous pouvons voir que :

- Le stégo-image généré par l'algorithme LSB est au format png et celui généré par l'algorithme F5 est au format jpg.
- La taille du stégo-image généré par l'algorithme LSB varie plus ou moins un peu si l'image de couverture est une image au format png. Si l'image est au format bmp (Bitmap), elle est compressée au format png et peut être réduite jusqu'à la moitié de sa taille initiale. Parcontre, si l'image de couverture est au format jpg, la taille du stégo-image augmente considérablement et peut atteindre jusqu'à 3 fois sa taille initiale, d'où il n'est pas vraimment recommandé d'utiliser ce format pour une image de couverture pour l'algorithme LSB, pour éviter que le stégo-image soit suspect (en effet, un stégo-image de plus de 8 Mo générée par une image de couverture de 2 Mo serait suspicieux), sauf pour des images de couverture inférieures à 2 Mo.
- L'algorithme F5 quant à lui, réduit considérablement la taille de l'image, par compression JPEG, si elle est au format png ou bmp, cependant, la taille varie plus ou moins un peu si elle est au format jpg.
- La technique LSB engendre un meilleur PSNR que l'algorithme F5, mais a le désavantage de concentrer les modifications au niveau des premiers pixels, contrairement à l'algorithme F5 (de manière pseudo-aléatoire), bien qu'en somme la différence entre le résultat obtenu et l'originale pour les deux techniques est indetectable visuellement.

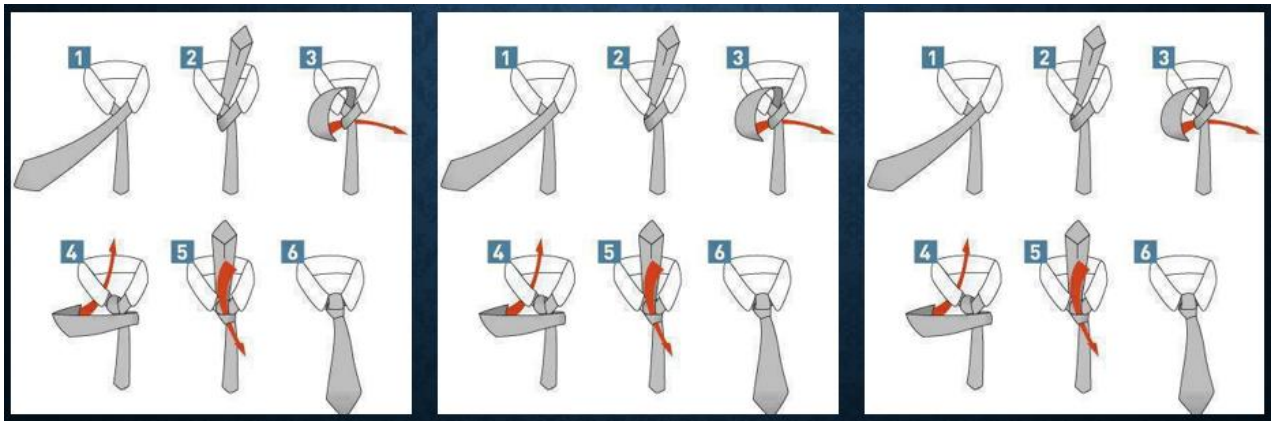


Figure 5.05 : *Comparaison visuelle entre l'image originale et les stégo-images LSB et F5*

Figure 5.06 : I*mage originale « Anniv.bmp »*

Figure 5.07 : *Stégo-image LSB « Anniv_LSB_encoded.png »*

Figure 5.08 : *Stégo-image F5 « Anniv_F5_encoded.jpg »*

Test 2 : Nous faisons ensuite varier la taille du message à cacher dans l'image de couverture pour voir l'évolution du PSNR pour les deux algorithmes. Voici le résultat obtenu :

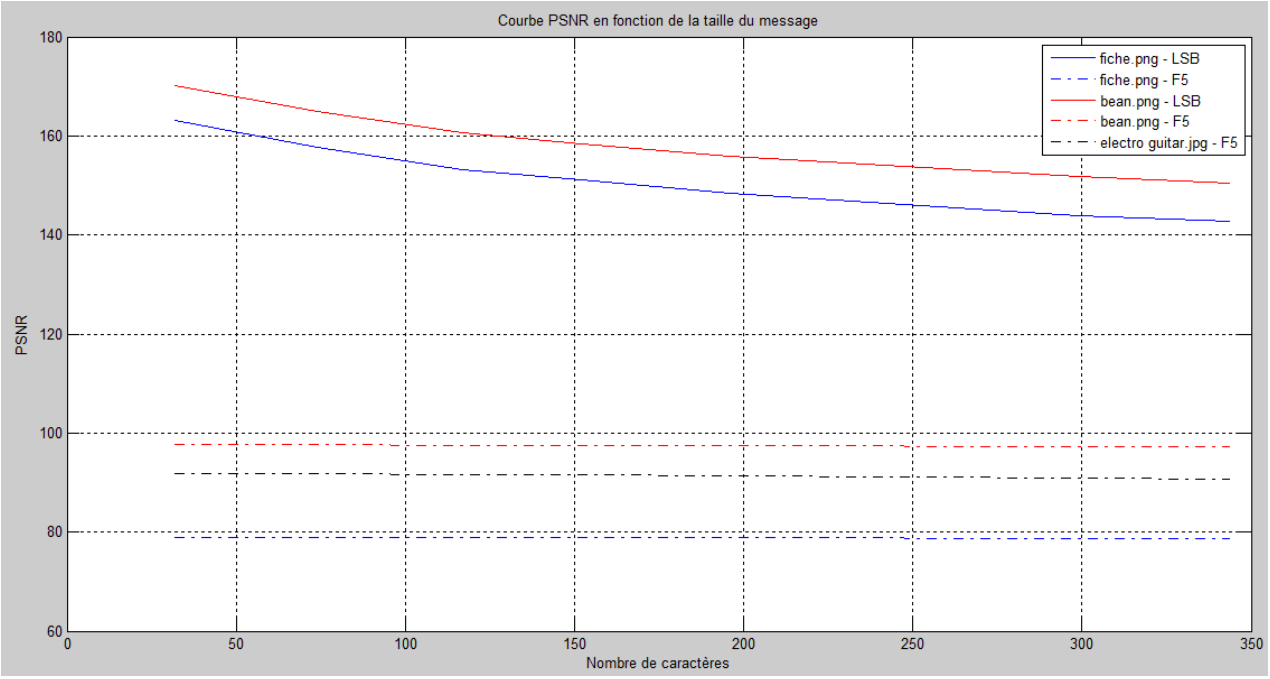


Figure 5.09 : *Evolution du PSNR en fonction de la taille du message caché*

Nous voyons que pour l'algorithme LSB, le PSNR décroit légèrement, mais visiblement en fonction du nombre de caractères du message. Ceci est dû au fait que plus ce nombre augmente, plus on modifie plus de pixels sur l'image, l'encodage étant effectué dans le domaine spatiale.

Par contre, la variation du PSNR pour l'algorithme F5 est très faible, voire négligeable, autour d'une valeur fixe. Ceci est dû probablement au fait que l'encodage se passe dans le domaine fréquentiel et le seul artéfact produit est une augmentation du nombre de coefficients à 0 ; cependant, cette caractéristique ressemblera simplement à une compression JPEG avec un facteur de qualité inférieur.

Test 3 : Ce test concerne l'encodage d'une image secrète dans une image da couverture avec la technique LSB. Notons que ces images devront être obligatoirement au format png ou au format bmp. C'est aussi pour ça que le choix d'images dans l'onglet « Encodage Image » de l'application ne se fait que dans la galerie d'images. L'utilisation de l'appareil photo n'est donc pas proposée car la photo capturée est souvent au format jpg. Pour la première partie de ce test, nous avons pris comme images à cacher des images au format png.

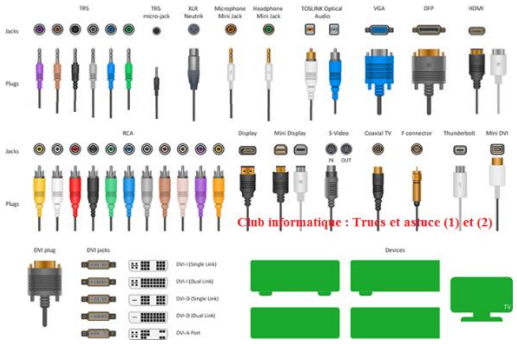
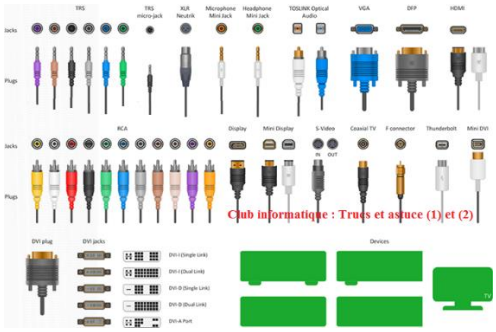
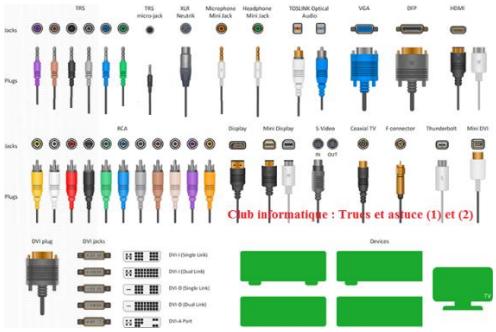
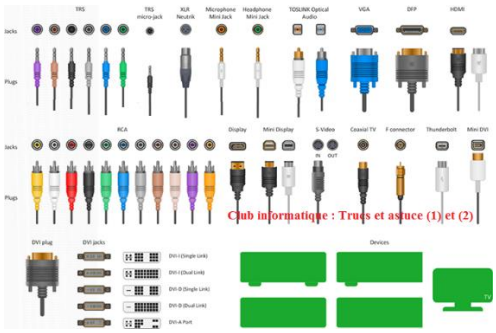
Images secrètes	Image de couverture	Stégo-images
Smile.png (128 x 128) 21.5 Ko 	Fiche.png (720 x 476) 128 Ko 	Fiche_LSB_encoded_smile.png (720 x 476) 180 Ko **PSNR** = 98.34 dB 
Bruce.png (256 x 256) 86.7 Ko 		Fiche_LSB_encoded_bruce.png (720 x 476) 365 Ko **PSNR** = 84.55 dB 
Lucy.png (692 x 900) 456 Ko 		Fiche_LSB_encoded_lucy.png (2160 x 1428) 2.56 Mo **PSNR** = non calculé 

Tableau 5.03 : *Résultat du test 3, première partie*

Remarquons que les images secrètes ont été parfaitement extraites à partir du stégo-image, étant donné que le calcul du PSNR entre chaque image extraite et avec l'image à cacher originale correspondante tend vers l'infini. Pour cette première partie du test, 3 images png, de tailles et de dimensions différentes ont été utilisées. Ce qu'il faut remarquer ici, c'est que la transparence est aussi gérée, en plus des couleurs RGB. En outre, le PSNR décroît assez rapidement suivant les dimensions de l'image secrète. Et finalement, il faut remarquer que pour encoder l'image Lucy.png de dimensions 692 x 900, de taille 456 Ko, dans l'image de couverture Fiche.png de dimensions 720 x 476, de taille 128 Ko, les dimensions du stégo-image généré ont été multipliées par 3, ce qui donne 2160 x 1428, contenant assez de pixels pour cacher l'image secrète. Le PSNR n'a donc pas pu être calculée car le stégo-image et l'image originale n'ont plus les mêmes dimensions.

En ce qui concerne la deuxième partie du test, l'image hôte et l'image secrète sont au format bmp.

Image secrète	**Image extraite**
Sapin.bmp (514 x 743) 1.09 Mo 	Sapin_extracted.png (514 x 743) 774 Ko PSNR tend vers l'infini 

Tableau 5.04 : *Comparaison entre l'image extraite et l'image originale*

Image de couverture originale	Stégo-image
Anniv.bmp (734 x 510) 1.07 Mo 	Anniv_sapin_encoded.png (2202 x 1530) 4.21 Mo 

Tableau 5.05 : *Comparaison entre l'image de couverture et le stégo-image*

Nous constatons que, bien que l'image secrète soit au format bmp et l'image extraite au format png, les deux images sont visuellement identiques, d'autant plus que le PSNR tend vers l'infini. Une autre différence est à propos de la taille qui a notablement diminuée pour l'image extraite grâce à la compression au format png (de 1.09 Mo à 774 Ko).

En ce qui concerne l'image de couverture et le stégo-image, les dimensions de l'image de couverture ont été multipliées par 3, afin de contenir suffisamment de pixels qui puissent cacher l'image secrète ((734 x 510) à (2202 x 1530)). La taille de l'image a elle aussi considérablement augmentée (de 1.07 Mo à 4.21 Mo).

Test 4 : Ce test est un test d'imbrication de la stéganographie. En effet, nous avons tenté de cacher une image dans une image de couverture, mais, l'image à cacher elle-même est un stégo-image dans lequel nous avons caché un message secret. Le résultat a été plutôt concluant. Nous avons obtenu un stégo-image qui cache une image, qui cache un texte. Le décodage de l'information consiste donc à faire une double extraction, d'abord une première fois pour extraire l'image secrète, ensuite une deuxième extraction sur l'image extraite pour décoder le message secret. Il faut seulement remarquer que l'imbrication pourra être faite autant de fois que la taille du stégo-image final généré ne soit pas suspecte. En effet, une taille d'image de 10 Mo commence déjà à éveiller les soupçons, cependant, cette valeur ne dépendra que du jugement de l'utilisateur. Le résultat de notre test est affiché dans le tableau 5.06.

Stégo-image cachant le texte chiffré	**Image de couverture principale**	**Stégo-image principal**
 Bean_LSB_encoded.png (400 x 545) 1.07 Mo	 Anniv.bmp (734 x 510) 1.07 Mo	 Anniv_bean_encoded.png (1468 x 1020) 1.99 Mo

Tableau 5.06 : *Imbrication de la stéganographie*

Voici le message caché :

"Mes comptes et mots de passe" :
RSitrakaH@yahoo.fr : RSitrakaYahooMDP0555
RSitrakaH@gmail.com : RSitrakaGMailMDP0555
Facebook : RSitrakaFacebookMDP0555

BaseDeDonneeEntreprise : AdminMDP0254PPnjl

OrdiPortablePersoAdmin : RSitrakaOrdiPersoMDP0555
OrdiPortableBureauAdmin : RSitrakaOrdiBureauMDP0555

Valise : 948
Carte ID : 1550245268425586865

Ce message a été tout d'abord encodé dans une image (bean.png), ce qui a généré un stégo-image « Bean_LSB_encoded.png ». Ensuite, cette nouvelle image a été à son tour encodée dans une autre image « anniv.bmp », ce qui a généré le stégo-image final « Anniv_bean_encoded.png ».

L'intérêt de cette opération est d'augmenter le niveau de protection de l'information. Ajouté au chiffrement de données AES, il réellement très difficile pour les tiers personnes de découvrir l'information cachée.

5.3 Bilan sur les deux algorithmes de l'application

Voici un petit tableau résumant les précisions à savoir sur les algorithmes de stéganographie utilisés par notre application.

LSB	F5
Disponible pour l'encodage de texte ou d'image dans une image de couverture	Disponible seulement pour l'encodage de texte dans une image
Facile à implémenter, peu couteux en temps de calcul	Lourd et complexe
PSNR décroit rapidement en fonction de la taille du message ou de l'image secrète	PSNR variant très peu autour d'une certaine valeur en fonction de la taille du message secret
Les premiers pixels de l'image de couverture sont modifiés pour insérer l'information (domaine spatiale)	L'information est insérée pseudo-aléatoirement dans les coefficients DCT de l'image lors de la compression jpg (domaine fréquentiel).
La taille du stégo-image augmente plus ou moins par rapport à l'image de couverture	La taille du stégo-image est généralement considérablement réduite grâce à la compression jpg de l'algorithme.
Le stégo-image généré est au format png	Le stégo-image généré est au format jpg
Les formats d'entrées recommandés sont : png, bmp ou parfois jpeg pour l'encodage de texte,	Les formats acceptés sont : png, bmp, et jpg

Tableau 5.07 : *Bilan sur les deux algorithmes de l'application*

5.4 Conclusion

Pour mieux cerner les différentes possibilités de l'application, ce chapitre s'est consacré à différents tests sur des images de tailles, de dimensions et de types différents. Il est possible par exemple d'effectuer une imbrication de la stéganographie en cachant un stégo-image cachant lui-même un texte, dans une image de couverture. Nous avons pu tirer quelques spécifications sur les deux algorithmes LSB et F5 de l'application, comme par exemple les types d'entrées acceptées et de sortie générée, ainsi que d'autres caractéristiques qui nous permettront de prédire le comportement de l'application lors d'une utilisation future.

CONCLUSION

Ce mémoire nous a permis de comprendre, non seulement l'intérêt de la stéganographie, mais aussi l'avantage et les contraintes liés au développement pour des systèmes embarqués. Nous avons pu alors développer une application mobile de stéganographie destinée à la plateforme Android, un système d'exploitation mobile populaire utilisé par certains téléphones portables, tablettes et autres appareils.

Dans un premier temps, nous avons présenté l'art de la stéganographie, notamment son intérêt par rapport à la cryptographie. Il s'avère que, contrairement à la cryptographie qui chiffre les données pour les rendre inintelligibles, mais cependant suspectes, la stéganographie offre l'avantage de dissimuler l'information dans un support innocent et banal, de telle sorte qu'aucune autre personne ne saurait suspecter l'existence même de l'information à l'intérieur. Ensuite une partie est consacrée à l'explication des techniques utilisées par l'application : la technique LSB, simple, facile à implémenter, ainsi que l'algorithme F5, un peu plus complexe ont été retenus. En outre, le message secret sera préalablement crypté à l'aide de la méthode de chiffrement standard AES, surtout connu pour sa grande résistance à toutes les attaques, pour renforcer la confidentialité de l'information. La troisième partie effectue une comparaison des meilleurs systèmes d'exploitation mobile actuels, et explique donc le choix de la plateforme Android comme plateforme de développement. Android de l'Open Handset Alliance, dont Google est un des pères fondateurs, est largement le système d'exploitation mobile le plus utilisé aujourd'hui, ce qui est sans doute expliqué par les nombreux avantages qu'il offre par rapport aux autres systèmes d'exploitation mobiles comme iOS de Apple, Windows Phone de Microsoft ou BlackBerry OS. La quatrième partie présente les outils de développement qui facilitent considérablement la programmation de l'application, les phases de conception, de développement et de l'implémentation de l'application. La dernière partie est réservée aux différents tests et à l'évaluation de la performance de l'application, dans le but de mieux cerner ses petites subtilités et d'anticiper de futures améliorations. Les cas d'utilisations de l'application n'aura alors de limites que celles de l'imagination de l'utilisateur.

L'application développée, au nom « MySteg » consiste donc à une application de stéganographie, qui implémente deux méthodes, la technique dite LSB s'effectuant dans le domaine spatial, et l'algorithme F5 s'effectuant dans le domaine fréquentiel. L'application permet de cacher un message secret, préalablement chiffré AES à l'aide d'un mot de passe fourni par l'utilisateur, dans

une image, soit avec la technique LSB ou F5, ou de cacher une image dans une image avec la technique LSB. Les opérations inverses sont bien évidemment possibles. Le stégo-image ou l'image extraite peut être partagé via MMS, WiFi, Bluetooth, email,…, selon les applications de partage installées sur le terminal de l'utilisateur. L'ergonomie de l'application a été faite de manière conviviale et intuitive ; mais dans le but d'améliorer l'application, d'autres algorithmes de stéganographie par exemple, en utilisant la transformation en ondelettes, peuvent être envisagés. En outre, nous pouvons aussi envisager d'utiliser un autre support de couverture, par exemple des fichiers audio, pour cacher l'information.

ANNEXE 1
TECHNIQUE DE MATRIX EMBEDDING

La technique de matrix embedding est une méthode de codage par syndrome, utilisant la théorie des codes correcteurs, en particulier les codes linéaires [9].

A1.1 Codes correcteurs d'erreurs, codes linéaires

Un code linéaire est un code correcteur d'erreurs structuré comme sous-espace vectoriel d'un corps fini. Ici, le corps utilisé est le corps de Galois à deux éléments $F_2 = \{0,1\}$.

Un code linéaire est défini par trois paramètres : [n, k, δ], avec n > k, où des messages de k bits sont transmis comme des mots de n bits. δ représente la distance minimale entre chaque mot du code. La distance utilisée est la distance de Hamming, définie pour tout $(\mathbf{x},\mathbf{y}) \in F_2^n \mathbf{x} F_2^n$ par :

$$\mathbf{d(x,y)} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{x_i} \oplus \mathbf{y_i} \qquad \text{(A1.01)}$$

Nous définissons de plus le poids de Hamming d'un vecteur x par d(x, 0), c'est à dire le nombre de 1 dans le vecteur.

Dans la théorie des codes correcteurs, deux entités souhaitent communiquer à travers un canal bruité. Pour cela, l'émetteur, qui désire envoyer un message $\mathbf{m} \in F_2^k$, transforme celui-ci en un mot de code $\mathbf{c} \in F_2^n$ par :

$$\mathbf{c = Gm} \qquad \text{(A1.02)}$$

Avec $\mathbf{G} \in \mathbf{M_{n,k}}$ la matrice dite ***génératrice*** du code linéaire. Le récepteur, qui reçoit un mot $\mathbf{c'} \in F_2^n$, calcule alors ce que l'on appelle le ***syndrome*** du code $\mathbf{s} \in F_2^{n-k}$ par :

$$\mathbf{s = Hc'} \qquad \text{(A1.03)}$$

Avec $\mathbf{H} \in \mathbf{M_{n-k,n}}$ la matrice dite ***de parité*** du code.

Si s = (0, ..., 0), alors c' = c et le message ne comporte pas d'erreur. Autrement, si s ≠ (0, ..., 0), alors le message comporte une ou plusieurs erreurs.

De plus, H étant une application de F_2^n dans F_2^{n-k}, plusieurs codes peuvent avoir le même syndrome s. On note alors C(s) l'ensemble des codes ayant s comme syndrome. Cet ensemble s'appelle la classe (coset) de s, et est défini par : $\mathbf{C(s) = \{c \in F_2^n : Hc = s\}}$. On appelle alors chef de classe (coset leader) l'élément de la classe qui a le poids de Hamming le plus faible.

A1.2 Utilisation des codes linéaires en stéganographie : matrix embedding

Dans notre utilisation des codes linéaires, ***le syndrome s représentera le message à transmettre***, et les données traversant le canal, **c'**, seront le support. Lors de l'extraction du message, le récepteur n'aura alors qu'à calculer le syndrome à l'aide de la matrice **H** pour obtenir le message caché. La difficulté se trouve ici du côté de l'émetteur [9].

En effet, le but est d'insérer un message $\mathbf{m} \in F_2^{n-k}$ dans un support $\mathbf{x} \in F_2^n$ en le modifiant le moins possible. Pour cela, le principe de la technique est de modifier **x** en **y** tel que :

$$\mathbf{Hy = m} \tag{A1.04}$$

Avec $\mathbf{H} \in \mathbf{M_{n-k,n}}$. Nous cherchons alors le vecteur $\mathbf{e} \in F_2^n$ qui modifie x en y, c'est-à-dire tel que :

$$\mathbf{y = x + e} \tag{A1.05}$$

Injectant l'équation 2.05 dans 2.04, on obtient :

$$\begin{aligned} &\mathbf{H(x+e) = m} \\ \Leftrightarrow\ &\mathbf{H(e) = m - H(x)} \end{aligned} \tag{A1.06}$$

Le vecteur **e** recherché est donc un code ayant comme syndrome **m − Hx**. Etant donné que nous cherchons à modifier le moins possible notre vecteur **x**, nous allons choisir le code ayant le poids de Hamming le plus faible : le vecteur optimal est le chef de la classe **C(m − Hx)**. On montre en fait que le nombre maximum de changement est inférieur au rayon de couverture **R** du code, car le vecteur **e** aura un poids de Hamming inférieur à **R**.

En général, la résolution de l'équation A1.06 est compliquée, et nécessite l'utilisation du pivot de Gauss, de complexité cubique sur le nombre de lignes de **H** (ici n-k). Cependant, l'utilisation de bons codes linéaires peut simplifier le problème, notamment avec des codes dont le rayon de couverture est faible.

A1.3 Exemple de matrix embedding : utilisation des codes de Hamming

Les codes de Hamming constituent une famille de codes linéaires permettant la détection et la correction d'une erreur si elle ne porte que sur un symbole du message. Leur distance minimale δ est égale à 3, et le rayon de couverture à 1 [9].

Ces codes sont paramétrés par un entier p : $n = 2^p - 1$ et $k = 2^p - 1 - p$. Les syndromes (donc les messages dans le cas de la stéganographie) sont donc de longueur $n - k = p$. La matrice de parité $\mathbf{H} \in \mathbf{M}_{p,2^p-1}$ est formée en plaçant sur chaque colonne la représentation en binaire des entiers de 1 à $2^p - 1$. Donc, si $m - Hx = He$ est un syndrome, alors (comme $m - Hx$ est de longueur p), il figure dans une des colonnes de H. Ainsi, e sera le vecteur formé d'un 1 à cette colonne, et de 0 ailleurs : un seul changement est nécessaire pour transformer notre x en y. Voici un exemple pour p = 3 :

Soit m = (1, 0, 1) le message à insérer dans x = (0, 1, 1, 1, 0, 0, 1).

La matrice H utilisée sera :

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Nous cherchons donc le vecteur $e = (e_1, e_2, e_3, e_4, e_5, e_6, e_7)$ tel que $H(x + e) = m$.

Pour cela, calculons $m - Hx$:

$$\mathbf{m - Hx} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

On a donc He = (1, 1, 1) qui correspond à la 7ème colonne de H, d'où e = (0, 0, 0, 0, 0, 0, 1).

Ainsi, le message transmis sera y = x + e = (0, 1, 1, 1, 0, 0, 0). Lors de l'extraction, le récepteur n'a alors plus qu'à calculer le syndrome.

$$\mathbf{Hy} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \end{pmatrix} = \mathbf{m}$$

Nous venons donc de voir que la technique de matrix embedding permet ***d'insérer p bits dans un support de $2^p - 1$ bits en effectuant au maximum*** (en effet, la technique engendre soit 0 soit 1 changement) ***un seul changement dans celui-ci***. Si cela parait à première vue très efficace, remarquons qu'une image de 512 x 512 pixels comporte 262144 pixels, donc 262144 bits potentiels pour servir de support. Si l'on veut appliquer la technique précédente, et n'effectuer qu'un seul changement dans l'image, le message doit être de longueur 18, car $2^{18} - 1 = 262143$ qui est < 262144. Cela représente une longueur relative de $\frac{\mathbf{18}}{\mathbf{262144}} = \mathbf{6,86.10^{-5}}$, ce qui est peu.

Une technique possible pour pallier à cette faiblesse est de diviser le message en plusieurs parties, et de dérouler la méthode avec chaque "morceau". Par exemple, pour p = 3, la technique nous permet d'insérer 3 bits dans un support de 7 bits, en n'effectuant au maximum qu'un seul changement. Ainsi, si l'on veut insérer 9 bits, il est nécessaire d'avoir un support de $2^9 - 1 = 511$ bits. Cependant, si l'on coupe le message en 3, alors on peut effectuer 3 fois l'insertion par matrice avec p = 3. Le support doit alors être de longueur $3 * 7 = 21$ bits, et celui-ci subira 3 changements. La longueur relative est maintenant de $\frac{\mathbf{9}}{\mathbf{21}} \approx \mathbf{0.24}$.

C'est de cette manière que sont implémentés la plupart des algorithmes de stéganographie utilisant la technique de matrix embedding, comme pour l'algorithme F5.

A1.4 Efficacité d'insertion

Nous allons maintenant nous intéresser à l'efficacité d'insertion de la technique de matrix embedding. ***L'efficacité d'insertion est définie comme le nombre de bits pouvant être insérés pour une modification dans le support*** [9].

Lorsque nous insérons p bits, dans un support de $2^p - 1$ bits, nous effectuons 0 ou 1 changement dans celui-ci. En effet, si m = Hx alors e = (0, ..., 0) ; le vecteur e représentant un nombre binaire

dans l'intervalle [0, $2^p - 1$], ce cas a une probabilité de $\frac{1}{2^p}$. Inversement, nous changeons 1 bits dans le support dans toutes les autres situations, donc avec une probabilité de $1-\frac{1}{2^p}$. Ainsi, le nombre moyen de changements pour insérer p bits est de : $0*\frac{1}{2^p}+1*(1-\frac{1}{2^p})=1-\frac{1}{2^p}=1-2^{-p}$

D'où le nombre de bits pouvant être insérés pour 1 seul changement, i.e. ***l'efficacité d'insertion*** :

$$\frac{p}{1-2^{-p}} \qquad \text{(A1.07)}$$

D'un autre côté, puisque nous insérons p bits dans $2^p - 1$ bits, ***la taille relative du message*** est de :

$$\frac{p}{2^p-1} \qquad \text{(A1.08)}$$

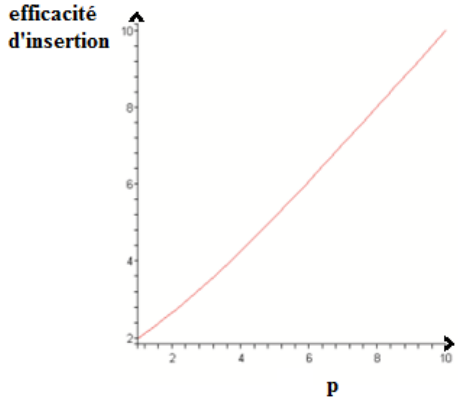


Figure A1.01 : *Evolution de l'efficacité d'insertion en fonction du paramètre p dans une technique de matrix embedding utilisant les codes de Hamming*

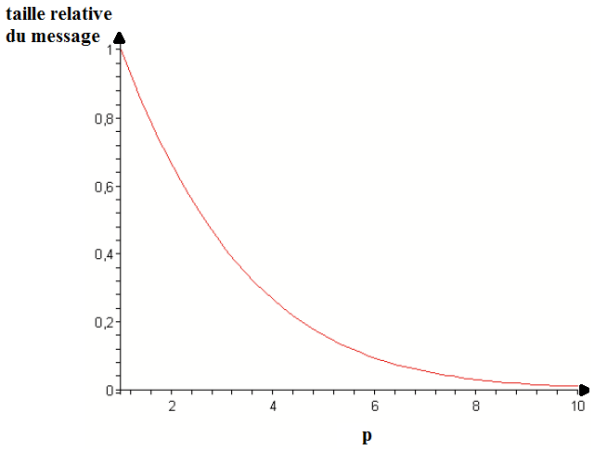


Figure A1.02 : *Evolution de la taille relative en fonction du paramètre p dans une technique de matrix embedding utilisant les codes de Hamming*

Choix du paramètre p : Nous pouvons donc voir que l'efficacité d'insertion ne dépend uniquement que du paramètre p utilisé. Dans la plupart des cas, comme expliqué à la fin du

paragraphe précédent, ce paramètre n'est pas directement égal à la taille du message à insérer. Ainsi, comment choisir ce paramètre de manière optimale ? En fait, nous allons voir que celui-ci ne dépend uniquement que de la taille relative du message à insérer. En effet, nous devons déterminer le nombre de fois que nous allons devoir diviser notre message afin de l'insérer. Soit K la longueur de notre message, N la longueur de notre support, $\alpha = \frac{K}{N}$ la taille relative du message et **a** le nombre de fois que l'on va devoir diviser ce dernier. Chaque changement engendrant une modification, nous devons choisir **a** le plus petit possible. Celui-ci doit en fait vérifier :

$$a(2^{\frac{K}{a}} - 1) \leq N < (a+1)(2^{\frac{K}{a+1}} - 1) \qquad \text{(A1.09)}$$

En effet, $L_a = (2^{\frac{K}{a}} - 1)$ est la longueur du support nécessaire pour un morceau de message de longueur $\frac{K}{a}$, donc aL_a est la longueur totale du support nécessaire. En divisant chaque membre de l'inégalité A1.10 par **a** ($a \neq 0$) on obtient :

$$2^{\frac{K}{a}} - 1 \leq \frac{N}{a} < \frac{(a+1)(2^{\frac{K}{a+1}} - 1)}{a} \Leftrightarrow \frac{a}{(a+1)(2^{\frac{K}{a+1}} - 1)} < \frac{a}{N} \leq \frac{1}{2^{\frac{K}{a}} - 1}$$

$$\Leftrightarrow \frac{K}{a} \frac{a}{(a+1)(2^{\frac{K}{a+1}} - 1)} < \frac{K}{a} \frac{a}{N} \leq \frac{K}{a} \frac{1}{2^{\frac{K}{a}} - 1}$$

$$\Leftrightarrow \underbrace{\frac{\frac{K}{(a+1)}}{(2^{\frac{K}{a+1}} - 1)}}_{\alpha_{a+1}} < \alpha \leq \underbrace{\frac{\frac{K}{a}}{(2^{\frac{K}{a}} - 1)}}_{\alpha_a} \qquad \text{(A1.10)}$$

α_a et α_{a+1} sont les tailles relatives pour une opération de matrix embedding avec respectivement $p = \frac{K}{a}$ et $p = \frac{K}{a+1}$.

Ainsi, le choix optimal pour le paramètre p se calcule à partir de la taille relative du message. Donc, l'efficacité d'insertion ne dépend elle aussi que de la taille relative de celui-ci.

ANNEXE 2

MARQUEURS JPEG COURANTS

Un fichier JPEG est constitué d'une séquence de segments commençant par un marqueur. Un marqueur se compose de la valeur 0xFF suivie d'un octet identifiant le type de marqueur. Certains marqueurs ne contiennent que ces deux octets ; d'autres sont suivis de deux octets spécifiant la taille en octets des données du segment. Cette taille inclut ces deux octets de taille mais pas ceux du marqueur.

Marqueurs JPEG courants

Abréviation	Valeur	Contenu	Nom	Commentaires
SOI	0xFFD8	*aucun*	Start Of Image	Premiers octets du fichier
SOF0	0xFFC0	*taille variable*	Start Of Frame (Baseline DCT)	Indique une image encodée par *baseline DCT*, et spécifie la largeur, la hauteur, le nombre de composantes et le sous-échantillonnage des composantes (par exemple 4:2:0).
SOF2	0xFFC2	*taille variable*	Start Of Frame (Progressive DCT)	Indique une image encodée par *progressive DCT*, et spécifie la largeur, la hauteur, le nombre de composantes et le sous-échantillonnage des composantes (par exemple 4:2:0).
DHT	0xFFC4	*taille variable*	Define Huffman Table(s)	Spécifie une ou plusieurs tables d'Huffman.
DQT	0xFFDB	*taille variable*	Define Quantization Table(s)	Spécifie une ou plusieurs tables de quantification.
DRI	0xFFDD	deux octets	Define Restart Interval	Spécifie l'intervalle entre les marqueurs RST*n*, en macroblocs. Ce marqueur est suivi de deux octets indiquant sa taille de sorte qu'il puisse être traité comme n'importe quel segment de taille variable.
SOS	0xFFDA	*taille variable*	Start Of Scan	Commence un parcours de haut en bas de l'image. Dans les encodages *baseline DCT*, il n'y a généralement qu'un seul parcours. Les images *progressive DCT* contiennent habituellement plusieurs parcours. Ce marqueur spécifie quelle tranche de données il contient et il est immédiatement suivi par des données codées entropiquement.
RST*n*	0xFFD0 … 0xFFD7	*aucun*	Restart	Inséré tous les *r* macroblocs, où *r* est l'intervalle DRI (cf. marqueur DRI). Il n'est pas utilisé s'il n'y a pas de marqueur DRI. Les trois bits de poids faible du code de marqueur varient en boucle de 0 à 7.
APP*n*	0xFFE*n*	*taille variable*	Application-specific	Ce marqueur permet d'inclure des informations qu'un programme de visualisation peut ignorer tout en restant capable de décoder l'image. Par exemple, un fichier JPEG Exif utilise un marqueur APP1 pour enregistrer des métadonnées, organisées selon une structure proche du formatage TIFF.
COM	0xFFFE	*taille variable*	Commentaire	Contient un commentaire textuel.
EOI	0xFFD9	*aucun*	End Of Image	Derniers octets du fichier

Tableau A2.01 : *Tableau des marqueurs JPEG courants*

ANNEXE 3

PROCEDURE SUBBYTES VIA S-BOX DU CHIFFREMENT AES

La procédure SubBytes du chiffrement AES peut alors être simplifiée à l'aide de l'utilisation de la S-box, au lieu d'utiliser les opérations matricielles. Voici comment remplacer l'octet 0x95 en utilisant le tableau de la S-box :

S(95)=42=0x2a

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure A3.01 : *Utilisation de la S-box de la procédure SubBytes*

ANNEXE 4

COMPLEMENTS SUR LE DEVELOPPEMENT DE L'APPLICATION

Voici quelques notions à savoir sur la programmation Android :

Figure A4.01 : *Configuration du terminal pour les anciennes versions d'Android*

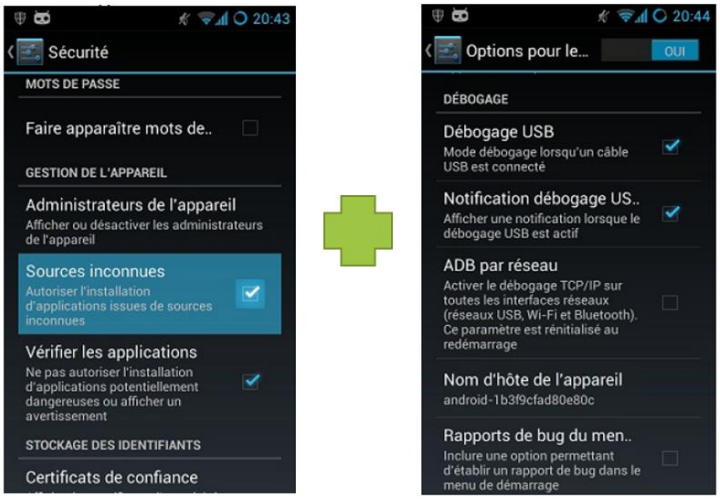


Figure A4.02 : *Configuration du terminal pour les nouvelles versions d'Android*

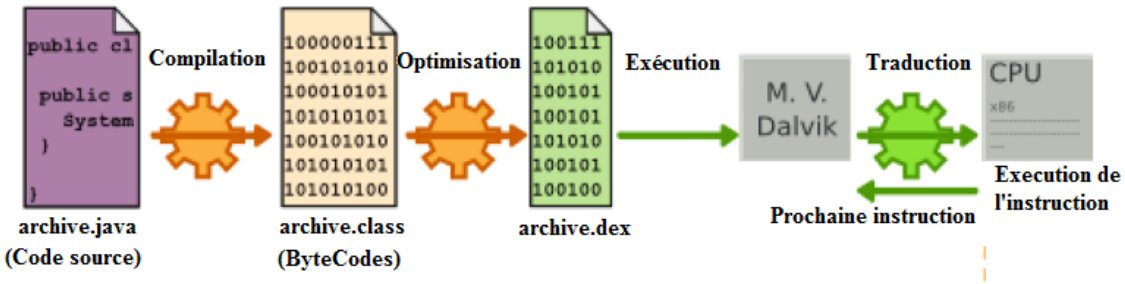


Figure A4.03 : *Processus de compilation d'un code Android*

Voici le script du fichier AndroidManifest.xml de l'application MySteg :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mg.espa.tco.stegano"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />

    <uses-feature android:name="android.hardware.camera"  android:required="true" />

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" >
    </uses-permission>

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
        <activity
            android:screenOrientation="portrait"
            android:name="mg.sitraka.espatco.main.SplashScreenActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar"
            android:noHistory="true" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:screenOrientation="portrait"
            android:name="mg.sitraka.espatco.main.MainLayout"
            android:label="@string/app_name">
        </activity>
        <activity
            android:screenOrientation="portrait"
            android:name="mg.sitraka.espatco.main.TextTab"
            android:label="@string/app_name" >
        </activity>
        <activity
            android:screenOrientation="portrait"
            android:name="mg.sitraka.espatco.main.PictureTab"
            android:label="@string/app_name" >
        </activity>
    </application>

</manifest>
```

BIBLIOGRAPHIE

[1] F.M.M. Bouyé, « *Application des codes correcteurs d'erreurs en stéganographie* », Thèse, Faculté des Sciences, Univ. Mohammed V Agdal, Rabat, 11 juillet 2012.

[2] D. Martins, « *Sécurité dans les réseaux de capteurs sans fil Stéganographie et réseaux de confiance* », Thèse,U.F.R. des Sciences et Techniques, Univ. Franche-Comt, 29 Nov 2010.

[3] G.J. Simmons, « *The prisoners' problem and the subliminal channel* », Crypto, pp. 51-67, 1983.

[4] C. Cachin, « *An information-theoretic model for steganography* », Inf. Comput., 192(1), pp. 41-56, 2004.

[5] B. Khaled, « *Approche par marquage pour l'évaluation de la qualité d'image dans les applications multimédias* », mémoire (INF6021) de maîtrise en informatique, Dép. Informatique et ingénierie, Univ. Québec en outaouais, Nov 2012.

[6] J. Pugliesi, C. Piovano, « *Le tatouage d'images ou watermarking* », Travail d'études, Licence d'informatique, Univ. Nice- Sophia Antipolis, juin 2004.

[7] A. Guilmain, « *la stéganographie au québec : lumière technique, ombres juridiques* », Lex Electronica, vol. 17.2, Automne/Fall 2012.

[8] H.A. Salas, « *La Steganographie Moderne : L'art De La Communication Secrète* », mémoire M2, Sciences et Techniques du Languedoc, Académie de Montpellier, Univ Montpellier II, 21 juin 2010.

[9] R. Watrigant, « *Utilisation des codes correcteurs d'erreurs en stéganographie : De l'algorithme F5 et sa stéganalyse aux codes à papier mouillé* », Licence 3 Mathématiques Informatique, Module Projet Informatique, Univ de Nîmes, A.U. 2008-2009.

[10] A. Westfeld, « *F5 - A Steganographic Algorithm* », High Capacity Despite Better Steganalysis, 1999.

[11] E. Incerti, « *Compression d'image. Algorithmes et standards* », Vuibert, 2003.

[12] D.A. Huffman, « *A method for the construction of minimum redundancy codes* », In Proc. IRE, volume 40, pp. 1098-1101, septembre 1952.

[13] T.E. Rakotondraina, « *Cryptographie* », Cours I3 – TCO, Dép. TCO.- E.S.P.A., A.U. : 2011-2012.

[14] P. Jeulin, « *Introduction Aux Techniques De Chiffrement Et De Securite, L'advanced Encryption Standard Ou AES* », Cours Cycle C, CNAM Limoges, A.U. : 2003-2004.

[15] Wikipedia, « *Comparison of mobile operating systems*», http://en.wikipedia.org/wiki/Comparison_of_mobile_operating_systems, 13 Fevrier 2015.

[16] Wikipedia, « *iOS (Apple)* », http://fr.wikipedia.org/wiki/IOS_(Apple), 27 février 2015.

[17] Wikipedia, « *Windows Phone* », http://fr.wikipedia.org/wiki/Windows_Phone, 11 janvier 2015.

[18] Wikipedia, « *BlackBerry* », http://fr.wikipedia.org/wiki/BlackBerry, 19 février 2015.

[19] F. Espiau, « *Créez des applications pour Android* », www.openclassrooms.com, 7 oct. 2013.

[20] J. Garcin, « *Statistiques Internet - Janvier 2015*», www.iwebyou.fr/actualites/statistiques-internet-janvier-2015, 26 janvier 2015.

[21] IDC, « *Press Release*», http://www.idc.com/getdoc.jsp?containerId=prUS25450615, 24 février 2015.

[22] M.L. Murphy, « *L'art du développement Android* », 2ème édition, Pearson, 2010.

[23] Y. Khmou, « *Peak Signal-to-Noise Ratio for RGB Images* », Matlab Central File Exchange, juillet 2012.

FICHE DE RENSEIGNEMENT

Nom **:** RAKOTOARISON

Prénoms **:** Sitraka Hasinarivo

Adresse **:** Lot II J 53 Ter Ivandry
Antananarivo 101

Téléphone **:** +261 33 03 122 42

E-mail : rakotoarison.sitraka@yahoo.fr

Titre du mémoire **:**

DEVELOPPEMENT D'APPLICATION MOBILE DE STEGANOGRAPHIE

Nombre de pages **:** 117

Nombre de tableaux : 29

Nombres de figures **:** 74

Directeur de mémoire : Monsieur RAKOTOMALALA Mamy Alain
rakotomamialain@yahoo.fr
+ 261 33 12 036 09

RESUME

Le présent mémoire nous a permis de connaître et de comprendre ce qu'est l'art de la stéganographie. L'objectif principal de ce travail est de développer une application mobile de stéganographie. La stéganographie est l'art et la science de la communication invisible, souvent confondue avec la cryptographie. C'est l'art de dissimuler une information dans une information, et donc en cachant l'existence même de l'information à transmettre. Notre application nommée « MySteg » permet de cacher un message secret préalablement chiffré AES ou une image secrète, dans une image hôte de couverture, à l'aide de deux techniques différentes : la méthode LSB, qui est simple et facile à implémenter, et l'algorithme F5, plus complexe et plus difficile.

MySteg a été développé pour le système d'exploitation mobile Android, le plus utilisé de tous actuellement. Bien que l'application soit déjà satisfaisante, elle peut être améliorée en implémentant d'autres algorithmes tels que la stéganographie par transformation en ondelettes ou tout simplement en ajoutant plus de fonctionnalités, par exemple cacher l'information dans un fichier audio.

Mots clés **:** Stéganographie, caché, imperceptibilité, crypté, Mobile

ABSTRACT

This memory allowed us to know and to understand what the art of steganography is. The main goal of this work is to develop a mobile steganography application. Steganography is the art and science of invisible communication, often confused with cryptography. It is the art of hiding information in other information, thus hiding the existence of the information to be communicated. Our application named "MySteg" allows hiding a secret message which is AES encrypted first or a secret image, into a host image as cover through two different technics : LSB method, which is simple and easy to implement, and F5 algorithm, more complex and harder.

MySteg was developed for Android operating system which is the most used mobile OS at this moment. However the application is already satisfying, it can be improved by implementing other algorithms such as Discrete Wavelet Transform steganography approach, or simply adding more functionalities such as hiding information into an audio file.

Keywords **:** Stéganography, hidden, imperceptibility, encrypted, Mobile