



Ingénieurs du numérique • Inspiring your digital future



Projet TOP 2019-2020

JOUONS À CACHE-CACHE

Encadrants : professeurs intervenant dans le module TOP

Groupe 11

Hugo BENAMEUR
Coralie DELARBRE
Niels TILCH

Table des matières

1	Introduction	4
1.1	Contexte du projet	4
1.2	Remerciements	4
2	État de l'art	5
2.1	La stéganographie, qu'est-ce que c'est ?	5
2.2	Premières traces	6
2.3	Stéganographie pure	7
2.4	La stéganographie de nos jours	8
2.4.1	Cacher du texte dans une page web	8
2.4.2	Cacher du texte dans un programme exécutable ou sur les parties libres du disque dur	8
2.4.3	Cacher du texte dans une image (.png ou .jpg)	8
Constitution d'une image	8	
Méthode	9	
2.4.4	Cacher une image dans une image	10
2.4.5	Stéganographie dans un fichier audio	10
2.5	Une méthode qui pose certains problèmes	11
2.6	Une utilisation démocratisée	11
3	Gestion de Projet	12
3.1	Charte de projet	12
3.1.1	Résumé	12
3.1.2	Cadrage	12
Finalité et importance du projet	12	
Contexte et hypothèses de départ	12	
Objectifs et résultats opérationnels	12	
3.1.3	Déroulement du projet	13
Organisation, ressources, budget	13	
Échéances, événements importants	13	
Risques et opportunités	13	
3.2	Analyse du projet	13
3.2.1	Objectifs SMART	13
3.2.2	Matrice SWOT	14
3.2.3	Work Breakdown Structure	14
3.2.4	Matrice RACI	15
3.2.5	Diagramme de Gantt	15
3.3	Compte-rendus de réunions	16
3.3.1	Réunion du 18 novembre 2019	16
Installation du logiciel	16	

Planifications des objectifs, les articles initiaux	16
3.3.2 Réunion du 25 novembre 2019	17
Accord sur le diagramme de Gantt	17
Avancement du codage	17
Outils à utiliser	17
3.3.3 Réunion du 26 novembre 2019	18
Changement de logiciel	18
Etape -1	18
3.3.4 Réunion du 27 novembre 2019	19
Conflits sur Gitlab	19
Etape -1	19
3.3.5 Réunion du 3 décembre 2019	20
Avancement global du projet	20
3.3.6 Réunion du 9 décembre 2019	21
Test de la fonction de l'étape -1	21
Test de la fonction de l'étape 0	21
Les fonctions de l'étape 1	21
3.3.7 Réunion du 16 décembre 2019	22
Tests et debugging des petites fonctions	22
Planification de la suite du projet	22
3.3.8 Réunion du 26 décembre 2019	23
Avancement global du projet	23
3.3.9 Réunion du 3 janvier 2020	24
Avancement du code	24
Avancement du rapport	24
Avancement de l'état de l'art	24
3.3.10 Réunion du 6 janvier 2020	25
Bilan du projet	25
Etape 2	25
4 Conception de l'algorithme	26
4.1 Explications	26
4.1.1 étape -1	26
4.1.2 étape 0	26
4.1.3 étape 1	26
4.1.4 étape 2	29
4.1.5 étape 2bis	30
4.1.6 étape 2ter	30
4.1.7 étape 3	31
4.2 Phases de tests	32
4.2.1 Images carrées de mêmes dimensions en format PNG	32
4.2.2 Images rectangulaires de mêmes dimensions en format PNG	32
4.2.3 Images en format JPEG	32
4.2.4 Images de tailles différentes et de format différent	33
4.2.5 Fonctionnement des différentes étapes	33
5 Annexe : Images des tests	35
5.1 Images carrées de mêmes dimensions en format PNG	35
5.2 Images rectangulaires de mêmes dimensions en format PNG	36
5.3 Images en format JPEG	37
5.4 Images de tailles différentes et de format différent	38
5.5 Fonctionnement des différentes étapes	39

6 Bilan du projet	42
6.1 Bilan global	42
6.2 Bilan individuel	42
6.2.1 Coralie	42
6.2.2 Hugo	43
6.2.3 Niels	43
6.3 Nombre d'heures par parties du projet	43
7 Bibliographie	44

Chapitre 1

Introduction

1.1 Contexte du projet

En partenariat avec Jean-Marie Moureaux, enseignant spécialiste de la stéganographie et du taillage d'image/vidéo à l'école, l'équipe pédagogique nous a proposé un projet dans le cadre du module de TOP sur la stéganographie et ses procédés. Le sujet est accompagné de deux articles scientifiques : l'un de Neil F. Johnson et Sushil Jajodia [1] et l'autre de Zhou Wang et son équipe [2]. Ce projet permettra d'ajouter de nouveaux procédés, plus ou moins efficaces en fonction des méthodes existantes et des contraintes données.

La stéganographie est un procédé consistant à dissimuler de l'information (image, texte, etc...) dans une image. Dans la majorité des cas, la stéganographie est largement plus utilisé que la cryptographie [1] du fait de la capacité de l'information à ne pas être reconnue en tant que telle par celui qui la récupère. De nombreuses recherches ont été faite depuis des siècles afin d'améliorer les techniques.

Le projet a débuté par la dissimulation d'une image dans une autre image, toutes deux de même taille. En vue d'avoir une récupération optimale de l'information, nous avons utilisé plusieurs équations de l'article de Zhou Wang [2] qui permettent de quantifier l'erreur entre l'image originale et l'image récupérée après lui avoir appliqué le programme. Enfin, nous avons étendu le programme afin de cacher n'importe quel type d'information en prenant en compte les marges d'erreurs autorisées.

Le projet est rendu sous la forme d'un rapport documentant l'ensemble de notre travail avec un programme sous le langage Scala.

1.2 Remerciements

Nous tenons à remercier vivement toutes celles et tous ceux qui nous ont aidé, de près ou de loin, à mener à bien ce projet.

Tout d'abord l'ensemble de l'équipe pédagogique du module de TOP, pour leur disponibilité et leurs réponses précieuses.

Nous remercions également Jean-Marie Moureaux pour avoir proposé ce sujet aussi intéressant qu'enrichissant, culturellement et professionnellement parlant.

Enfin, un grand merci à nos collègues étudiants apprentis et non apprentis pour l'esprit d'entraide dont ils ont fait preuve pour répondre à nos questions et nous guider vers le bon chemin.

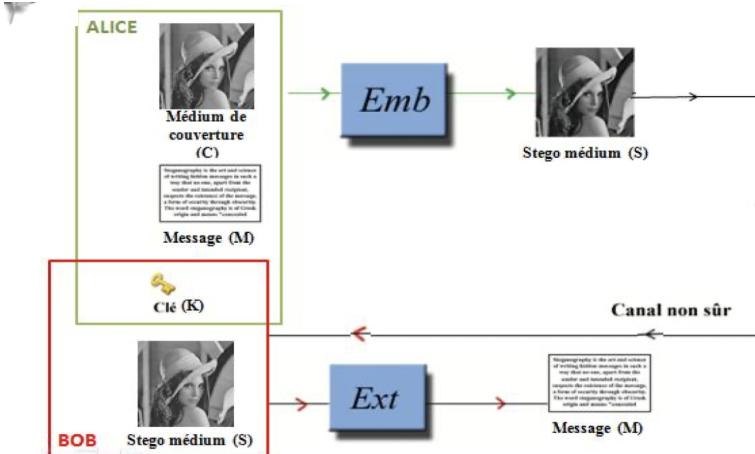
Chapitre 2

État de l'art

2.1 La stéganographie, qu'est-ce que c'est ?

La stéganographie a pour but de faire passer une information sans qu'elle ne soit visible. Cela consiste à cacher le contenu d'un message là où la cryptographie se contenterait d'en cacher le sens. On peut donc affirmer que la stéganographie est une méthode plus discrète car nous n'avons pas conscience de détenir une information clé lorsque l'on intercepte un message de ce type.

D'après l'étymologie du mot stéganographie, celui-ci signifie écriture dissimulée. Cette méthode modifie donc le contenu du support sans en modifier son aspect. Par exemple, dans notre projet, l'image support reste identique avant et après avoir caché la seconde dans celle-ci. On appelle ce support « médium de couverture » lorsqu'il ne contient pas encore l'information à cacher puis « stégo-médium » une fois le message inséré.



R. Watrigant. Utilisation des codes correcteurs d'erreurs en stéganographie : De l'algorithme F5 et sa stéganalyse aux codes à papier mouillé, Rapport de projet en L3, Nîmes, France, 2009. 11

FIGURE 2.1 – Fonctionnement de la stéganographie

La stéganographie et la cryptographie sont des méthodes qui remontent à l'Antiquité. Utilisées ensemble elles offrent une sécurité maximale lors de l'envoi d'une information secrète. Nous allons maintenant découvrir comment la stéganographie s'est mise en place au fil des années.

2.2 Premières traces

La première apparition de la stéganographie a été relevée dans Histoire de Hérodote vers 445 avant J-C. En effet, dans cette œuvre on trouve deux récits reflétant bien l'utilisation de cette méthode. Dans la première anecdote, Histée souhaite envoyer une missive à Aristagoras afin de lui ordonner de se soulever contre son roi Darius malgré une grande surveillance des routes. Pour cela, il fit raser la tête de l'un de ses esclaves afin d'y tatouer son message. Une fois que ses cheveux avaient repoussés, Histée l'envoya dans la ville où se trouvait Aristagoras avec pour ordre de demander à se faire raser la tête dès son arrivée.

Dans la seconde anecdote, Démarate racla la surface des tablettes de cire afin d'écrire directement son message sur le bois. Un fois celui-ci écrit, il recouvrit de nouveau les tablettes avec de la cire afin de les envoyer sans que l'alerte qu'il avait écrite ne soit visible.

Les Hommes n'ont pas cessé d'utiliser la stéganographie au cours de l'histoire, on peut noter l'apparition de techniques diverses toujours plus novatrices. Durant l'Antiquité, par exemple, les chinois écrivaient leurs messages sur de la soie qu'ils roulaient en boule et recouvriraient ensuite de cire pour que leurs messagers puissent les avaler. Giovanni Porta quant à lui découvrit comment cacher ses messages dans des oeufs durs. A l'aide d'une encre constituée d'alun et de vinaigre, il écrivait sur un oeuf. L'écriture traversait la coquille pour "s'imprimer" sur l'oeuf lui-même. Vu de l'extérieur, l'oeuf semblait quelconque cependant une fois pelé on découvrirait le message écrit auparavant.

Durant les guerres, envoyer des messages sans que l'ennemi ne s'en aperçoive est quelque chose de très recherché. Là encore, on dénombre beaucoup de méthodes différentes comme des microfilms cachés sous les timbres ou encore le micropoint. Cela consiste à réduire une image jusqu'à ce qu'elle soit de la taille d'un point puis de la cacher dans un texte banal.

Enfin, une des autres techniques désormais très connue consiste à écrire avec de l'encre "invisible" comme du lait, du jus de citron ou même de l'urine. Pour décoder le message il suffit de le placer au dessus d'une flamme comme le montre le schéma ci-dessous.

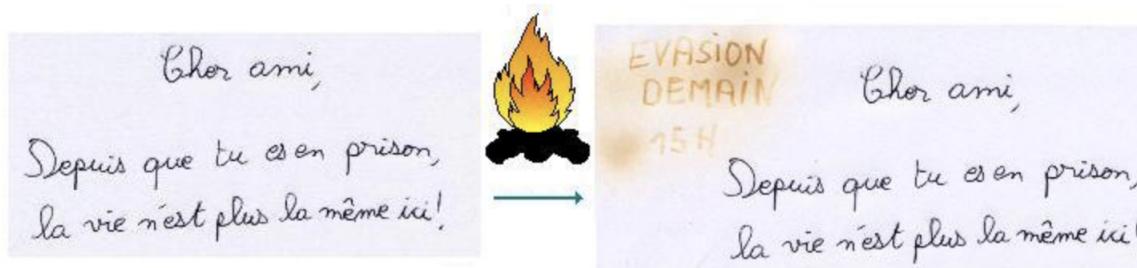


FIGURE 2.2 – Fonctionnement de l'encre invisible

2.3 Stéganographie pure

Les différents procédés cités précédemment ne sont pas de la stéganographie pure car ils modifient le support en lui-même. L'acrostiche quand à lui est une des méthodes qui se sert du contenu déjà présent dans le médium de couverture pour y cacher le message souhaité. Cela consiste à prendre, par exemple, la première lettre de chaque vers pour décrypter le code. En voici un exemple :

Paradis des monuments
Attend petits et grands
Rêve des touristes et des enfants
Illuminé par tous les temps
Sur la tour Eiffel c'est géant

FIGURE 2.3 – Exemple d'acrostiche

Sur ce même principe il existe donc le mésostiche, le télestiche ou encore l'acroteleuton. De nombreuses méthodes sont issues de cette idée, il suffit seulement d'un peu d'imagination et de partager avec la personne souhaitée la façon dont le message à été réalisé : message à lire une ligne sur deux, de bas en haut, etc...

L'Abbé Jean de Trithème de son côté créa une sorte d'alphabet à partir de bout de phrase. L'association des lettres ressemblait alors à une prière ce qui semblait tout à fait normal pour un abbé.

A = dans les cieux	N = en paradis
B = à tout jamais	O = toujours
C = un monde sans fin	P = dans la divinité
D = en une infinité	Q = dans la déité
E = à perpétuité	R = dans la félicité
F = sempiternel	S = dans son règne
G = durable	T = dans son royaume
H = sans cesse	U-V-W = dans la bonté
I-J = irrévocablement	X = dans la magnificence
K = éternellement	Y = au trône
L = dans la gloire	Z = en toute éternité
M = dans la lumière	

FIGURE 2.4 – Code inventé par l'abbé Jean de Trithème

2.4 La stéganographie de nos jours

Au fur et à mesure des siècles les technologies ont évolué, laissant place à des méthodes de stéganographie toujours plus précises et discrètes. Bien évidemment, la liste des procédés précédents n'est pas exhaustive mais nous avons choisi de ne pas en dire trop sur les anciennes méthodes afin d'expliquer au mieux celles qui concerne notre projet ou qui sont en relation avec l'informatique.

2.4.1 Cacher du texte dans une page web

Une page web est écrite grâce au langage HTML qui ne tient pas compte du nombre d'espaces. En effet, qu'importe l'intervalle entre chaque mot cela sera toujours retranscrit de la même manière. On peut donc se servir de cela pour y cacher un message secret en se mettant d'accord au préalable sur un alphabet.

Nous allons prendre comme exemple l'alphabet le plus simple, le nombre d'espaces correspondra à la place de la lettre dans l'alphabet classique (A=1, B=2, etc...).

Voici le message incluant le code :

Quel_____est_____le_____mot_____caché_____dans_____cette_____phrase ?
20 5 12 5 3 15 13

Il apparaîtra de cette manière :

Quel est le mot caché dans cette phrase ?

FIGURE 2.5 – Exemple pour cacher du texte dans une page Web

Le mot caché est TELECOM, afin de le découvrir il suffira de regarder le code source de la page web et de compter les espaces. C'est une méthode plutôt facile mais assez fragile car certains logiciels peuvent perdre les données et donc le message.

2.4.2 Cacher du texte dans un programme exécutable ou sur les parties libres du disque dur

Pour cacher du texte dans un programme exécutable, il suffit d'ajouter le message dans une partie du programme qui ne sera jamais atteinte. Afin de voir le secret dissimulé, il faudra donc ouvrir le code du bon fichier exécutable et le lire.

Pour écrire un message dans une partie libre du disque dur, il est nécessaire de le chiffrer pour qu'il passe inaperçu. Il faut aussi faire attention à ne pas écrire dans son index que les données ont été érites à cet endroit. Enfin, pour retrouver le message il suffira de connaître son adresse.

2.4.3 Cacher du texte dans une image (.png ou .jpg)

Constitution d'une image

Une image est constituée de pixels dont le nombre varie en fonction de sa qualité mais aussi de sa taille. Chacun d'eux est représenté par une matrice de quatre chiffres allant de 0 à 255 représentant respectivement la transparence, l'intensité rouge, verte et bleue comme le montre le schéma ci-après. Comme on peut le voir un pixel mesure donc 32 bits.

Pour l'ordinateur, une image est donc une matrice de matrices de nombres binaires. Dans le cas des images en noir et blanc, les trois valeurs R,G et B sont les mêmes.

8	8	8	8
Alpha	Red	Green	Blue
31	30	29	28
27	26	25	24
23	22	21	20
19	18	17	16
15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

FIGURE 2.6 – Schéma d’explication de la composition des bits

Comme expliqué précédemment, chaque pixel est composé de 4 nombres codés sur 8 bits. Les 4 premiers bits (de gauche à droite) sont appelés “bits de poids fort” tandis que les 4 derniers sont appelés “bits de poids faible”.

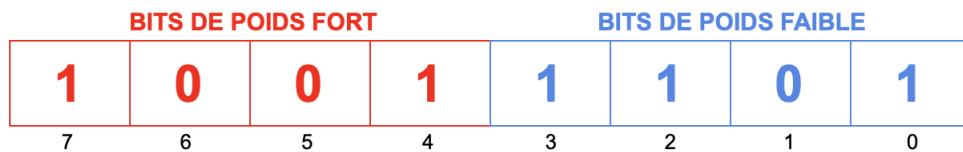


FIGURE 2.7 – Schéma d’explication du poids des bits

Ceci s’explique par le fait que plus la modification d’un bit est située à gauche plus cela sera visible à l’oeil nu. En effet, si l’on prend l’exemple ci-dessus, si l’on modifie le bit numéro 6 cela aura un plus grand impact que si l’on modifiait le bit numéro 3 (le nombre initial est 157, si l’on modifie le bit numéro 6 il devient 221 alors que si l’on modifie le numéro 3 il devient 149.). On utilise donc cette propriété dans les méthodes suivantes.

Méthode

On peut donc se servir des pixels d’une image pour y cacher un texte à l’intérieur. En effet, il suffit de remplacer un certain nombre de bit de poids faible par les différents caractères du message à cacher. Pour cela il est donc nécessaire de convertir notre message en binaire. Voici un exemple :

Message à cacher : “100111000110100”

Exemple de pixels disponibles : 10111101 11010111 10111111 11111111
00111001 11110101 10111110 10110101

Nous décidons de cacher le messages dans les deux derniers bits de chaque octet. Cela donnera donc :

Pixels contenant le message : 10111110 11010101 10111111 11111110
00111000 11110111 10111101 10110100

FIGURE 2.8 – Exemple pour cacher du texte dans une image

La modification du pixel n’entraînant qu’une variation minimale au niveau des couleurs, il est impossible pour l’œil humain de se rendre compte qu’un changement a été effectué. Dans cet exemple, nous avons modifié uniquement deux bits par octet ce qui entraîne une variation maximale de 3 sur 255 nuances disponibles soit une nuance infime.

2.4.4 Cacher une image dans une image

Le processus le plus utilisé pour cacher une image dans une image est la méthode Least Significant Bit (LSB). Elle consiste à remplacer les bits de poids faible du médium de couverture par les bits de poids fort de l'image à cacher.

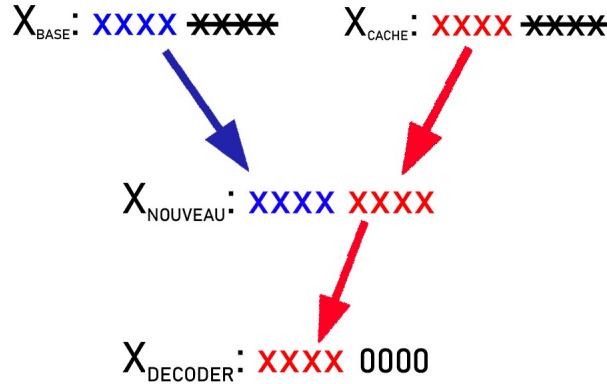


FIGURE 2.9 – Explications de la méthode LSB

Cette méthode a pour avantages d'être facile à coder ainsi que rapide ; cependant il y a une grande perte d'information ce qui peut parfois poser problème.

Dans les documents fournis, il est évoqué la méthode Discrete Cosine Transform (DCT) qui permet une dissimulation de l'image plus complexe en prenant en compte la luminosité, la rotation de l'image, etc... Cette transformée est par exemple utilisée dans l'algorithme de Koch et Zhao. Cependant nous avons décidé de ne pas en dire plus à ce sujet car nous ne sommes pas sûrs d'avoir réellement compris comment cela marchait.

Durant nos recherches, nous avons trouvé un processus ressemblant à une fusion entre la méthode LSB et celle pour cacher du texte dans une image. En effet, pour cacher du texte dans une image, on choisit un nombre de pixels de poids faible à modifier dans lesquels on placera les différents bits des pixels de l'image à cacher. La différence entre ce processus et la méthode LSB est que dans ce cas l'entièreté des pixels sont cachés dans l'image et non uniquement les bits de poids forts. Cependant il est nécessaire que le médium de couverture soit beaucoup plus grand que l'image à cacher (cela dépendra du nombre de bits modifiés choisi). On aura alors beaucoup plus de chance d'avoir une image précise et nette en décodant le stégo-médium. Attention, il est possible de choisir plus de bits et de ne pas se limiter aux bits de poids faible cependant cela reviendra à prendre plus de risques car l'image sera de moins en moins bien cachée.

2.4.5 Stéganographie dans un fichier audio

Il existe différents processus de stéganographie sur des fichiers audios.

- **Low Bit Encoding (LBE)** : est similaire à la LSB, il s'agit de cacher les bits correspondant à ce que l'information "secrète" (un message par exemple) à la place des bits de poids faible du fichier audio.
- **Spread Spectrum Encoding** : (qui date des années 30) consiste, elle, à ajouter un bruit aléatoire (qui contient l'élément à cacher) à la mélodie originale du fichier audio. Le message à cacher est transformé en bruit qui est étalé sur un large spectre de fréquences. L'étalement spectral est en effet une solution de cryptage et de camouflage efficace qui ajoute de la sécurité au processus et peut même être réalisé sur le signal final.
- **Echo Data Hiding** : lorsqu'un son est émis, il est suivi de son écho [18]. Cet écho peut rester imperceptible s'il dure jusqu'à 1ms au maximum. Il est donc possible de cacher de l'informa-

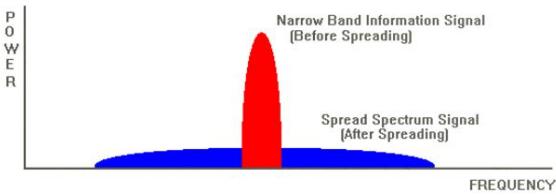


FIGURE 2.10 – Explications de la méthode spread spectrum

tion dans cet écart entre le son et son écho, mais aussi de créer un écho tel qu'il contienne l'information à cacher, tout en restant imperceptible.

- **Masque de perception :** il est aussi possible de cacher un son derrière un son plus puissant mais de même intensité. Ainsi, le son caché restera imperceptible mais fera partie du fichier.

La stéganographie sur les fichiers audio présente des avantages par rapport à celle sur les images. La taille des fichiers audio est d'abord plus importante que celle des images ; on peut donc théoriquement y cacher davantage d'informations, en utilisant une des nombreuses techniques de dissimulations (plus variées que pour les images).

2.5 Une méthode qui pose certains problèmes

La stéganographie est une méthode très sécurisée... Si l'on ne connaît pas la manière dont à été caché le message ! En effet, une fois que l'on sait comment cela a été réalisé il devient beaucoup plus simple de décoder le message. De plus, il faut désormais faire attention à ne pas utiliser des méthodes déjà très connues car de nombreux algorithmes existent afin de les détecter et donc de décoder le message aisément.

Lorsque l'on utilise des images pour la stéganographie, il est important de faire attention au format utilisé. Par exemple, si l'on prend un fichier BMP pour y cacher un message il est important de ne pas le convertir par la suite en JPG car cela entraînerait une perte de qualité voire la disparition du message due à la compression subie par l'image.

2.6 Une utilisation démocratisée

La stéganographie est désormais utilisée à d'autres fins que l'envoi de messages. En effet, elle est par exemple utilisée dans le commerce pour identifier l'origine d'un produit ou pour que celui-ci ne soit pas dupliqué de manière illégale. Cela s'appelle le tatouage numérique ou watermarking. Cela consiste donc à marquer un document de façon indélébile et invisible afin de pouvoir le reconnaître par la suite. Cette méthode est appliquée sur les DVD "zonés", achetés aux USA ces derniers ne peuvent, en théorie, pas être lus par des lecteurs Européen. Cela ne fonctionne pas toujours car il est nécessaire que le lecteur utilisé vérifie le marquage du DVD. Enfin ces algorithmes de tatouages numériques possèdent plusieurs contraintes : ils doivent être invisibles, difficilement supprimables, résister aux transformations comme les compressions ou l'impression et enfin doivent être reconnaissables.

Malheureusement, la stéganographie est aussi utilisée à des fins malveillantes par des hackeurs. Grâce à cela, ils infiltreront donc des ordinateurs en propageant des fichiers à priori sains. C'est le cas du ransomware SyncCrypt : les pirates ont caché un fichier zip dans une image d'apparence inoffensive et l'ont envoyée par mail. Les utilisateurs ont alors reçu un lien qui semblait officiel (ex : administration) ce qui les a poussés à cliquer dessus. Cela a lancé un programme JavaScript qui a extrait le contenu du zip. Des logiciels malveillants ont alors crypté le contenu de certains fichiers sensibles de l'ordinateur de l'utilisateur puis une rançon d'environ 450\$ leur a été demandée pour pouvoir récupérer la clé de déchiffrement.

Chapitre 3

Gestion de Projet

3.1 Charte de projet

3.1.1 Résumé

Le sujet porte sur la stéganographie. Il nous est demandé de faire des recherches sur les enjeux liés à cette pratique, les objectifs, mais aussi les différentes méthodes qui existent et leur évolution. L'activité principale au cours du projet sera de cacher des messages dans des images par différents moyens.

3.1.2 Cadrage

Finalité et importance du projet

Ce projet s'inscrit dans la validation du module de TOP pour l'année universitaire 2019-2020. De plus, il s'agit là du premier projet des trois années du cursus ; la bonne mise en place des éléments de base de la gestion de projet et la capacité à rédiger un rapport de qualité ont pour but de lancer de manière concrète la formation d'ingénieur.

Contexte et hypothèses de départ

La maîtrise de la gestion de version avec Git, ShareLatex et du langage Scala sont les principales compétences requises pour assurer le bon déroulement du projet.

Objectifs et résultats opérationnels

L'objectif est d'implémenter plusieurs méthodes permettant de cacher des messages de formes diverses (son, texte, image...) dans une image. Il est attendu que l'équipe livre le code source du projet et un rapport rédigé depuis Latex. Celui-ci devra comporter :

- Un état de l'art présentant la stéganographie, ses diverses méthodes et leur évolution
- Une explication de la conception de l'algorithme
- Un compte-rendu des difficultés rencontrées et des solutions mises en place
- Des preuves que le programme fonctionne à l'aide des outils vus en cours de TOP
- Des éléments de gestion de projet

Il nous est demandé en outre d'utiliser le projet Git qui nous a été affecté sur le GitLab de l'école.

3.1.3 Déroulement du projet

Organisation, ressources, budget

Étant donné le niveau modeste des trois membres de l'équipe projet pour ce qui concerne le codage en Scala et l'algorithmique, l'écriture du code, la réalisation des livrables de gestion de projet et des divers documents sera équitablement répartie entre les membres.

Pour réaliser ce projet, nous ne nécessitons d'aucun budget, seulement des développeurs et de leur machine respective. Une API munie de deux fonctions nous est en outre fournie pour faciliter la manipulation des images dans l'environnement Scala.

Échéances, événements importants

DATES	ÉCHÉANCES
Mercredi 6 novembre 2019	Lancement du projet
Mercredi 8 janvier 2020	Dernier délai pour transmettre les livrables sur Arche
Lundi 13 janvier 2020	Soutenance de projet en trinôme

Risques et opportunités

- Risques
 - S'enfermer dans un rythme trop lent qui nous ferait prendre du retard avant l'enchaînement des partiels.
 - Vouloir faire tout le travail en groupe. La clé pourrait être de se répartir les tâches de façon à avancer sur plusieurs points en simultané et accélérer le développement du projet.
- Opportunités
 - Apprendre des erreurs que l'on pourra faire, des difficultés rencontrées pour parfaire notre gestion de projets plus volumineux.

3.2 Analyse du projet

3.2.1 Objectifs SMART

Spécifique	Nous avons fixé des objectifs ciblés en fonction de chaque étape.
Mesurable	Nous obtenons un résultat après chaque étape, on peut donc vérifier si le but souhaité est atteint.
Accepté	Souhaitant valider notre année, nous sommes prêts à donner le meilleur de nous même.
Réalisable	Les objectifs peuvent être atteints sinon ils ne seraient pas demandé par les professeurs.
Temps	Nous disposons d'une date limite.

3.2.2 Matrice SWOT

Nous avons décidé de ne réaliser qu'une seule matrice SWOT pour l'ensemble du groupe car nous pensons avoir des profils assez similaires. Cela serait donc inutile de créer 3 fois le même tableau.

Strengths :	Weaknesses :
- équipe motivée - cohésion de groupe	- niveau modeste en programmation, débutants en Scala - priorité accordée à d'autres échéances importantes pendant une partie du projet (partiels)
Opportunities :	Threats :
- entraide entre différents groupe lors de grandes difficultés - utilisation d'internet	- temps - matériel défaillant - emplois du temps peu conciliables

3.2.3 Work Breakdown Structure

Lot 1 : Logiciel de sténographie

Lot 1.1 : Étape -1 : Cacher une image dans une autre image de taille identique avec la méthode LSB

Lot 1.1.1 : Convertir l'image et l'ensemble des pixels en matrice de matrice de nombres hexadécimal

Lot 1.1.2 : Sélectionner les bons bits de l'image à cacher et de l'image de base

Lot 1.2 : Étape 0 : Retrouver l'image cachée par la méthode LSB

Lot 1.2.1 :Retrouver les bits de l'image cachée

Lot 1.2.2 : Reconstituer l'image

Lot 1.3 : Étape 1 : Déterminer la qualité de l'image ressortie avec plusieurs indices mathématiques

Lot 1.3.1 : Calculer EQM

Lot 1.3.2 : Calculer SSIM

Lot 1.3.3 : Calculer MSSIM

Lot 1.4 : Étape 2 : Trouver une méthode de masquage permettant de garantir un augmentation de la qualité

Lot 1.4.1 : Récupérer le bon nombre de bits à modifier dans chaque pixel

Lot 1.4.2 : Modifier uniquement les bits choisis

Lot 1.5 : Étape 2bis : Déterminer la quantité d'information disponible à mettre avec une marge d'erreur autorisée

Lot 1.6 : Étape 2ter : Être capable d'interpréter un message/ binaire dans une image (MIME)

Lot 1.7 : Étape 3 : Trouver d'autres méthodes permettant de cacher un message

Lot 1.8 : Réaliser les tests finaux

Lot 2 : Rapport à rendre

Lot 2.1 : GDP

Lot 2.2 : État de l'art

Lot 2.3 : Rédaction

3.2.4 Matrice RACI

Lots	Livrables ou tâches	Niels	Coralie	Hugo
Lot 1	Logiciel de sténographie			
Lot 1.1	Etape 1 : Cacher une image dans une autre image de taille identique avec la méthode LSB	R A	R A	A R
Lot 1.1.1	Convertir l'image et l'ensemble des pixels en matrice de nombres hexadécimal	R	R	R
Lot 1.1.2	Sélectionner les bons bits de l'image à cacher et de l'image de base	A	R	R
Lot 1.2	Etape 0 : Retrouver l'image cachée par la méthode LSB	A A	R R	R R
Lot 1.2.1	Retrouver les bits de l'image cachée	A	R	R
Lot 1.2.2	Reconstituer l'image	A	R	R
Lot 1.3	Etape 1 : Déterminer la qualité de l'image reçoit avec plusieurs indices mathématiques	A R R	R A A	A R R
Lot 1.3.1	Calculer EQM	A	R	A
Lot 1.3.2	Calculer SSIM	R	A	R
Lot 1.3.3	Calculer MSSIM	R	A	R
Lot 1.4	Etape 2 : Trouver une méthode de masquage permettant de garantir un augmentation de la qualité	A A	R R	R R
Lot 1.4.1	Récupérer le bon nombre de bits à modifier dans chaque pixel	A	R	R
Lot 1.4.2	Modifier uniquement les bits choisis	A	R	R
Lot 1.5	Etape 2bis : Déterminer la quantité d'information disponible à mettre avec une marge d'erreur autorisée	R	A	A
Lot 1.6	Etape 2ter : Être capable d'interpréter un message/ binaire dans une image (MIME)	R	A	A
Lot 1.7	Etape 3 : Trouver d'autres méthodes permettant de cacher une message/ un message	R	A	A
Lot 1.8	Réaliser les tests finaux	A	R	R
Lot 2	Logiciel de sténographie			
Lot 2.1	GDP	R	R	R
Lot 2.2	État de l'art	R	R	R
Lot 2.3	Rédaction	R	R	R
R	Réalisateur			
A	Approbateur			
C	Consultant			
I	Informé			

FIGURE 3.1 – Matrice RACI réalisée par l'équipe

3.2.5 Diagramme de Gantt

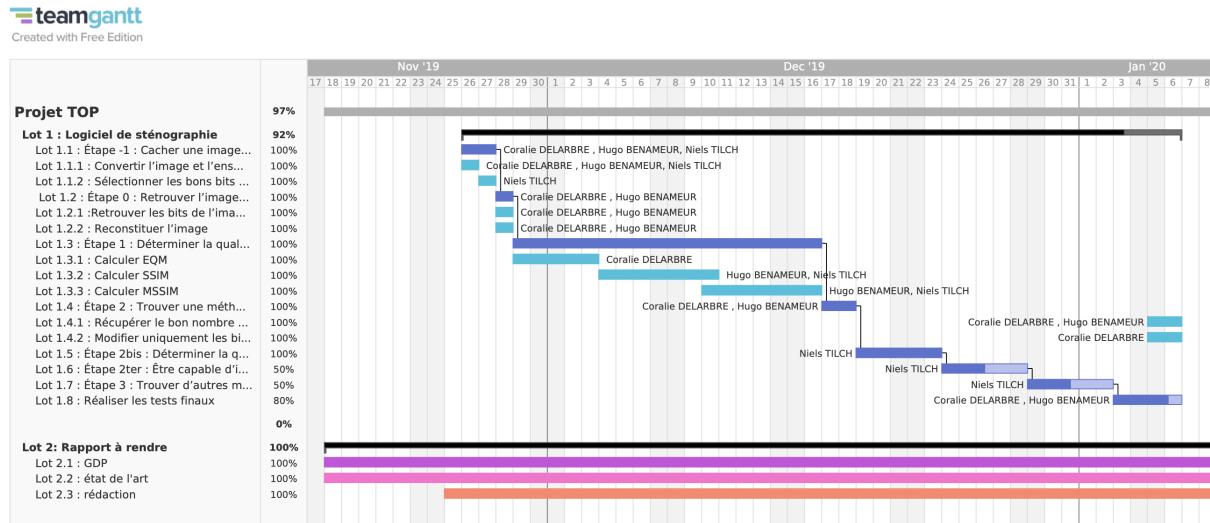


FIGURE 3.2 – Diagramme de Gantt réalisé par l'équipe

3.3 Compte-rendus de réunions

3.3.1 Réunion du 18 novembre 2019

Date : Lundi 18 novembre 2019

Durée : 30 min

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Heure de début : 12h

Ordre du jour :

- Résumé des articles initiaux du sujet
- Planification des objectifs de la gestion de projet
- Installation du logiciel

Installation du logiciel

La réunion s'est tournée principalement autour de l'installation et l'utilisation du logiciel IntelliJ et sur la possibilité de migrer vers Eclipse avec l'API que le sujet nous a fournie.

L'installation de l'ensemble du logiciel (avec API) est quasiment terminée et l'étape codage va pouvoir commencer d'ici la semaine prochaine après avoir bien analysé les articles initiaux et ce que le sujet nous demande.

Planifications des objectifs, les articles initiaux

Le rendez-vous tient compte qu'il est nécessaire de faire un résumé au propre d'ici la semaine prochaine et de trouver (optionnel) d'autres documents concernant le sujet.

L'ensemble des personnes est d'accord sur le fait de réfléchir ensemble à un diagramme de Gantt en fonction des étapes à fournir pour le bon déroulement du projet.

La matrice RACI sera faite dès lors que les objectifs seront clairs pour l'ensemble de l'équipe.

L'ordre du jour étant épuisé, la réunion s'est terminée à 12 heures 30.

Prochaine réunion : Lundi 25 novembre 2019

Pour la prochaine fois :

Hugo Benameur	Coralie Delarbre	Niels Tilch
- Lecture et résumé articles - Installer intelliJ	- Lecture et résumé articles	- Lecture et résumé articles

3.3.2 Réunion du 25 novembre 2019

Date : Lundi 25 novembre 2019

Durée : 30 min

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Heure de début : 12h

Ordre du jour :

- Validation du diagramme de Gantt
- Méthodes à utiliser pour réussir les étapes -1 et 0

Accord sur le diagramme de Gantt

Durant notre réunion nous avons discuté des différents diagrammes de Gantt proposés par chaque membre de l'équipe.

Nous sommes donc tous tombés d'accord sur le fait qu'il était important d'avancer dans la gestion de projet mais aussi sur les différentes étapes du code.

Lorsque le diagramme de Gantt final a été établi, nous l'avons retranscrit à l'aide du logiciel en ligne TeamGantt.

Avancement du codage

Comme expliqué ci-dessus il était important pour nous d'avancer sur le code demandé. Nous avons donc réfléchi ensemble aux méthodes qui permettraient de résoudre les étapes -1 et 0.

Outils à utiliser

Enfin cette réunion a permis de rappeler qu'il était important d'utiliser les différents logiciels proposés par TELECOM comme Gitlab et ShareLaTeX. En effet, nous n'avons pas l'habitude d'utiliser ces plateformes pourtant essentielles durant ce projet.

Le temps ne permettant pas de travailler sur les étapes -1 et 0, nous avons décidé de terminer cette réunion.

Prochaine réunion : Mardi 26 novembre 2019

Pour la prochaine fois :

Hugo Benameur	Coralie Delarbre	Niels Tilch
Réaliser l'étape -1 et 0	Réaliser l'étape -1 et 0	Réaliser l'étape -1 et 0

3.3.3 Réunion du 26 novembre 2019

Date : Mardi 26 novembre 2019

Heure de début : 18h

Durée : 1h

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Ordre du jour :

- Changement de logiciel
- Réalisation de l'étapes -1

Changement de logiciel

Après discussion entre les membres du groupe, nous avons décidé d'utiliser IntelliJ IDEA plutôt qu'Eclipse qui nous paraissait plus compliqué.

La majorité de cette réunion consistait donc à installer ce nouveau logiciel.

Etape -1

Une fois le logiciel installé, nous avons donc mis en oeuvre les méthodes discutées la veille. Cependant ce n'était pas aussi simple que prévu. Nous nous sommes donc laissé une soirée pour y réfléchir.

Souhaitant réfléchir à de nouvelles méthodes d'action, nous avons donc décidé de programmer une nouvelle réunion le lendemain.

Prochaine réunion : Mercredi 27 novembre 2019

Pour la prochaine fois :

Hugo Benameur	Coralie Delarbre	Niels Tilch
Régler le problème rencontré	Régler le problème rencontré	Régler le problème rencontré

3.3.4 Réunion du 27 novembre 2019

Date : Mercredi 27 novembre 2019

Heure de début : 12h

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Durée : 1h

Ordre du jour :

- Régler le problème survenu à l'étape -1

Conflits sur Gitlab

N'ayant pas l'habitude d'utiliser Gitlab, nous devions régler des conflits survenus lors de nos différents push. Cela a été une première étape essentielle durant cette réunion.

Etape -1

Après avoir réfléchi la veille, nous avons discuté des différentes solutions afin de régler notre problème. Chaque membre de l'équipe a donc exposé son idée puis nous avons choisi la plus pertinente.

L'ordre du jour étant épuisé, nous avons donc mis fin à cette réunion.

Prochaine réunion : Mardi 3 décembre 2019

Pour la prochaine fois :

Hugo Benameur	Coralie Delarbre	Niels Tilch
Réfléchir à l'étape 0	Réfléchir à l'étape 0	Mettre en place la solution proposée

3.3.5 Réunion du 3 décembre 2019

Date : Mardi 3 décembre 2019

Heure de début : 18h

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Durée : 30 min

Ordre du jour :

- Avancement global du projet

Avancement global du projet

Lors de cette réunion de routine, nous avons fait le point sur l'état d'avancement du projet et de la gestion de projet.

- Concernant la gestion de projet, le GANTT a été lancé et est mis à jour régulièrement, la matrice RACI ainsi que le WBS sont constitués, de même que la charte de projet.
- Les étapes -1 et 0 ont été terminées pour ce qui concerne le code.

Une mise au point a aussi été faite concernant la gestion du projet avec Git. Le respect des branches et de leur rôle n'est pas encore acquis mais nous avons rappelé l'importance de bien séparer les phases de développement du programme (dev, test, master).

L'ordre du jour étant épuisé, la réunion s'est terminée à 18h30.

Prochaine réunion : Lundi 9 décembre 2019

Pour la prochaine fois :

Pour tout le monde, lecture et résumé du 2e article pour pouvoir avancer sur l'étape 1 (maîtrise nécessaire des méthodes de calcul décrites)

3.3.6 Réunion du 9 décembre 2019

Date : Lundi 9 décembre 2019

Durée : 1h 15min

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Heure de début : 18 heures

Ordre du jour :

- Test des fonction de l'étape -1 et 0
- Résumé du deuxième article du Sujet
- La création des différentes fonctions de l'étape 1

Test de la fonction de l'étape -1

Nous avons tout d'abord expliqué en détails le fonctionnement du programme avant de commencer les tests. Des problèmes sont survenus lors de la visualisation de l'image avec les pixels des bords de droite et d'en bas. Ainsi, les modifications ont été apportées sur les indices de la matrice de l'image. Après l'utilisation de quelques images de taille similaires, nous avons conclu du bon fonctionnement de la fonction.

Il est remarqué par Hugo que certaines modifications de la fonction seront nécessaire pour l'étape 2.

Test de la fonction de l'étape 0

On a effectué le même procédé que l'étape -1. Aucun problème n'est apparu lors de l'exécution de la fonction.

Nous avons remarqué des différences entre l'image que l'on a cachée et l'image que nous avons récupérée. Ainsi, nous avons compris l'importance de l'étape 1 permettant la quantification des différences entre les deux images.

Les fonctions de l'étape 1

Après la lecture du deuxième article, Hugo a résumé l'ensemble des calculs à faire pour la quantification de qualité de l'image entre l'entrée et la sortie.

La liste des fonctions donnée par Hugo est la suivante : les écart-types, niveau de gris, la moyenne, SSIM, MSSIM, EQM. Il est précisé par Niels que l'ensemble des calculs se font avec une image en noir et blanc.

Il est décidé de la répartition des fonctions à programmer (cf. TO DO LIST).

L'ordre du jour étant épuisé, la réunion s'est terminée à 19 heures 15.

Prochaine réunion : Lundi 16 décembre 2019

Pour la prochaine fois (liste des fonctions à faire) :

Hugo Benameur	Coralie Delarbre	Niels Tilch
- Conversion de l'image en noir et blanc - La fonction moyenne	- Les écart-types - La fonction EQM	- La fonction MSSIM - La fonction SSIM

3.3.7 Réunion du 16 décembre 2019

Date : Lundi 16 décembre 2019

Heure de début : 18h

Durée : 1h

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Ordre du jour :

- Tests et debugging des petites fonctions
- Planification de la suite du projet

Tests et debugging des petites fonctions

Depuis la dernière réunion, les petites fonctions nécessaires pour faire tourner la méthode SSIM ont été codées. Cependant, lors des différents tests réalisés, nous nous rendons compte de certains bugs. Cette séance est principalement dédiée à la résolution de ces bugs.

Planification de la suite du projet

Nous avons en parallèle cherché des informations sur les différents moyens de réaliser les étapes 2, 2bis et 2ter. Nous avons fait un état des lieux de ce qu'il reste à faire (à savoir principalement la mise à jour des éléments de gestion de projet, état de l'art et étapes 2). Il a été convenu que chacun pourrait avancer à sa guise lors de cette semaine de fin de partiels et de Noël et qu'une réunion serait prévue le 26 décembre.

L'ordre du jour étant épuisé, la réunion s'est terminée à 19h.

Prochaine réunion : Jeudi 26 décembre 2019

Pour la prochaine fois : Pour tout le monde : avancer si possible dans les tâches évoquées plus haut.

3.3.8 Réunion du 26 décembre 2019

Date : Jeudi 26 décembre 2019

Heure de début : 10h15

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Durée : 1h

Ordre du jour :

- Avancement global du projet

Avancement global du projet

Nous avons fait le point sur l'état d'avancement du projet et de la gestion de projet.

- Concernant la gestion de projet, nous allons mettre dans les jours qui arrivent un maximum de livrables sur le dépôt Git. Chacun s'est vu attribuer des compte-rendus de réunion à mettre au propre.
- Niels ayant une idée plus claire que Coralie et Hugo sur la façon de coder les étapes 2, il a été décidé qu'il avancerait sur le codage afin de permettre aux autres membres du groupe d'avoir une idée plus claire de la marche à suivre. Pendant ce temps, Coralie et Hugo avancent sur la rédaction du rapport dont l'introduction a été commencée par Niels.

Étant donné que la date de rendu approche, nous avons décidé qu'à présent, une personne travaillera sur le code pendant que les deux autres font de la gestion de projet ou rédigent l'état de l'art.

L'ordre du jour étant épuisé, la réunion s'est terminée à 11h15.

Prochaine réunion : Vendredi 3 janvier 2020

Pour la prochaine fois :

Hugo Benameur	Coralie Delarbre	Niels Tilch
- Avancer dans la rédaction du rapport	Avancer dans la rédaction du rapport	Avancer sur les étapes 2

Pour tout le monde : Chacun doit rechercher des articles pour l'état de l'art et prouver (ou expliquer précisément) les programmes qu'il a écrits

3.3.9 Réunion du 3 janvier 2020

Date : Vendredi 3 janvier 2020

Durée : 1h

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Heure de début : 11h15

Ordre du jour :

- Avancement du code
- Avancement du rapport
- Avancement de l'état de l'art

Avancement du code

Pendant la semaine, Niels a codé l'étape 2bis qui est prête pour la phase de tests. Il a aussi écrit les programmes de l'étape 2ter permettant de cacher un message dans une image et a implémenté une méthode de cryptage et décryptage pour l'étape 3 du projet. Nous avons maintenant pris la décision de laisser Hugo et Coralie s'occuper de la phase de tests. L'objectif étant d'automatiser les tests pour gagner du temps, nous écrirons un programme pour réaliser les tests que l'on veut à l'étape que l'on veut. Pour cela, nous constituerais une petite banque d'images de tailles et types variés qui permettront de tester plusieurs configurations possibles. Coralie écrira le programme global et constituera la banque d'images, et Hugo codera les tests possibles sur chaque étape.

Avancement du rapport

Concernant le rapport, Hugo a réalisé en LaTeX le modèle du rapport (page de titre, structure, bibliographie) et y a ajouté le début d'introduction rédigé par Niels, ainsi que les explications des étapes -1, 0 et une partie de l'étape 1. Les compte-rendus des réunions antérieures sont à jour, mais le Gantt doit être actualisé.

Avancement de l'état de l'art

De nouveaux articles ont été trouvés par les différents membres du groupe. Coralie est chargée de les synthétisés afin de pouvoir les mettre dans l'état de l'art.

L'ordre du jour étant épousé, la réunion s'est terminée à 12h15.

Prochaine réunion : Lundi 6 janvier 2020

Pour la prochaine fois :

Hugo Benameur	Coralie Delarbre	Niels Tilch
Commencer le programme de tests finaux	Commencer le programme de tests finaux	Continuer les étapes 2, 2bis et 2ter

3.3.10 Réunion du 6 janvier 2020

Date : Lundi 6 janvier 2020

Heure de début : 14h00

Présents :

- Hugo Benameur
- Coralie Delarbre
- Niels Tilch

Durée : 2h

Ordre du jour :

- Bilan du projet

Bilan du projet

Durant cette réunion, l'ensemble de l'équipe a donné son avis sur le déroulement et l'avancée de notre projet. Cela a donc permis d'écrire le bilan final. Nous en avons aussi profité pour relire le rapport afin de déceler les fautes et le valider.

Etape 2

Pendant que nous discutions, nous nous sommes rendus compte que nous savions comment résoudre l'étape 2. Nous nous sommes donc penchés sur le sujet afin de pouvoir répondre à cette question. Une fois le code fait, il nous reste à remplir cette partie dans le rapport.

Une fois l'ensemble du travail terminé, nous avons arrêté cette réunion.

Chapitre 4

Conception de l'algorithme

4.1 Explications

La grande majorité des programmes que nous avons écrits sont formés d'une double boucle for. Dans ces cas, les pixels de l'image sont simplement balayés entièrement et la boucle s'arrête une fois le balayage terminé.

Il est rappelé que les matrices que nous manipulons pour les images sont des tableaux de tableaux avec pour coefficients des mots de 32 bits de cette forme :



FIGURE 4.1 – Modélisation d'un pixel

4.1.1 étape -1

cachée :

Cette fonction prend en entrée les matrices de l'image à cacher et de l'image dans laquelle on va cacher quelque chose, et renvoie l'image stéganographiée correspondant à l'image de base dans laquelle on a caché l'autre image. La forme *pixel* & *0xf0f0f0f0* nous permet par exemple de garder uniquement les 4 bits de poids fort pour le codage de chaque information contenue dans le pixel (transparence, rouge, vert, bleu).

Dans cette étape, cette méthode nous permet de sélectionner les 4 bits de poids forts de l'image à cacher ainsi que ceux de l'image principale et de les "regrouper" pour former les 8 bits de l'image stéganographiée pour chaque information contenue dans les pixels.

4.1.2 étape 0

retrouve :

De la même manière qu'à l'étape précédente, en reprenant l'image stéganographiée, les 4 bits de poids faible de chaque information contenue dans les pixels sont sélectionnés puis décalés à gauche de 4 rangs et viennent ensuite former les 4 bits de poids fort qui formeront l'image décodée.

4.1.3 étape 1

EQM :

Pour l'étape 1, nous avons d'abord implémenté la méthode de l'erreur quadratique moyenne (EQM) pour déterminer la perte de qualité du décodage par la méthode LSB. D'après l'article [2], l'EQM entre

deux images données se calcule en sommant sur la longueur et la largeur l'écart entre les valeurs d'un même pixel, élevé à une puissance n. La formule générale que nous avons utilisée est la suivante :

$$EQM = \left(\sum_{i=0}^{Longeur-1} \sum_{j=0}^{Largeur-1} (x_{i,j} - y_{i,j})^n \right)^{1/n}$$

Avec :

- $x_{i,j}$ (respectivement $y_{i,j}$) : luminance du pixel de coordonnées i, j de la zone x (resp. y)

decompoHexa :

Ce petit programme est utile dans la conversion en niveau de gris qui arrive juste après. Il prend en argument une chaîne de caractères qui représente un mot en hexadécimal, et en renvoie la conversion en décimal. Cette chaîne est convertie en décimal à l'aide de la méthode “Integer.parseInt([String],[Int])” qui permet de convertir une chaîne de caractères en un nombre dans la base que l'on souhaite.

nivGris :

Cette fonction prend en entrée le tableau d'une image et renvoie un tableau contenant les lumières de chacun de ses pixels. L'objectif de ce programme est de rendre une image couleur en image susceptible de fonctionner avec la méthode SSIM décrite dans l'article [2], c'est-à-dire la convertir en noir et blanc. Le calcul de luminance est fait à partir de la formule de la recommandation 709 de l'industrie audiovisuelle :

$$Luminance = 0,2126 * Rouge + 0,7152 * Vert + 0,0722 * Bleu.$$

Le programme balaye les pixels un par un, calcule la luminance du pixel, et la stocke dans la nouvelle matrice de mêmes dimensions que l'image, mais qui ne contiendra que des flottants. Cette matrice est retournée à l'issue du programme.

calculs :

Afin de calculer la valeur de perte de qualité par la méthode SSIM, il nous faut calculer la luminance moyenne et l'écart-type entre les valeurs de luminance d'une matrice d'image donnée. Ce programme prend en argument une matrice de flottants et renvoie un doublet de flottants de la forme (moyenne,écart-type). Pour ce qui est de la méthode utilisée dans le code, il s'agit d'un balayage de l'image grâce à une double boucle for. Pour le calcul de la moyenne, nous avons réalisé une simple moyenne arithmétique, alors que le calcul de l'écart-type a été réalisé selon la formule :

$$\sigma_x = \frac{1}{N} * \left(\sum_{i=0}^{Longeur-1} \sum_{j=0}^{Largeur-1} (x_{i,j} - \mu_x)^2 \right)$$

sigmaXY :

Ce programme prend en arguments deux tableaux correspondant à la même zone sur deux images X et Y, il a pour objectif de calculer le σ_{xy} décrit dans l'article [2] par la formule suivante :

$$\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_X)(y_i - \mu_Y)$$

Avec :

- N : nombre de pixels dans la zone
- x_i (respectivement y_i) : luminance du pixel i de la zone x (resp. y)
- μ_X (resp. μ_Y) : luminance moyenne de la zone sur l'image X (resp. Y)

MSSIM :

La fonction MSSIM est une fonction plus complexe que EQM permettant de normaliser l'erreur entre deux images et prenant "quantitativement" en compte la perception de l'oeil humain. Il y a deux parties dans la fonction MSSIM, la fonction principale faisant la moyenne (MSSIM) des calculs de SSIM et la fonction SSIM faisant le calcul d'erreur entre deux matrices d'images.

Le calcul de MSSIM se fait à partir d'une image en niveau de gris : une conversion en niveau de gris est nécessaire au début de ce programme et se fait grâce à la fonction nivGris décrite plus haut.

Dans l'article [1], il est expliqué que l'oeil observe l'image en plusieurs "zones" ; c'est-à-dire que nous sommes incapables de regarder une image dans sa globalité, mais nous nous concentrons tour à tour sur des parties plus petites. Ainsi, pour observer la moindre différence, l'oeil va essayer de détecter les erreurs sur des mêmes endroits. Ces "zones" peuvent être représentées comme des sous parties de l'image ou bien des sous-images.

Afin de créer ces sous-images, nous avons introduit le taux de division : c'est une variable prenant en compte deux paramètres dx et dy sous forme d'un tableau à 2 coefficients $[dx, dy]$. C'est grâce à ce taux de division que l'on va créer les sous-images ou plus précisément les sous-matrices dans le programme.

On prend une matrice de taille $m * n$.

Tout d'abord, l'algorithme s'occupe des premières sous-matrices en haut à gauche de taille $\lfloor m/dx \rfloor * \lfloor n/dy \rfloor$ (Figure 4.2)

Dans le cas où la division de m par dx ne donne pas une valeur entière, cela voudrait dire qu'il y a un reste non nul, des pixels n'ont donc pas été comptés dans le calcul. De la même manière, on crée les sous-matrices à droite puis en bas (Figures 4.3 et 4.4).

Il reste ainsi encore une sous matrice si les deux divisions n'ont pas données de valeur entière (Figure 4.5).

Le taux de division (que nous avons défini à 8) permet de donner un choix arbitraire de la manière dont l'utilisateur regarde la photo mais ne représente pas en soi une version exacte de son observation : c'est seulement une approximation car nous voulons éviter que le calcul prenne en compte un pixel plusieurs fois.

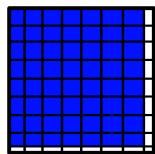


FIGURE 4.2 – Premières matrices détecté

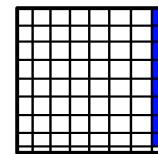


FIGURE 4.3 – Reste des sous-matrice à droite

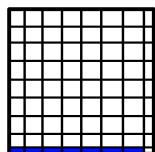


FIGURE 4.4 – Reste des sous-matrice en bas

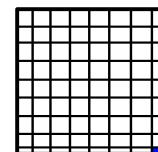


FIGURE 4.5 – Dernière sous-matrice

subArrayDim2 :

Ce programme prend en entrée la matrice d'une image déjà convertie en nuances de gris, l'abscisse minimale de la zone à extraire, l'abscisse maximale, l'ordonnée minimale et l'ordonnée maximale. Elle renvoie la matrice de la zone demandée, délimitée par les bornes en argument et correspondant à la sous matrice de la zone demandée. Ce programme servira pour calculer MSSIM.

SSIM :

Ce programme prend en argument les matrices converties en niveau de gris de la zone des deux images que l'on souhaite comparer, elle renvoie ensuite la valeur de perte de qualité SSIM pour la zone demandée.

D'après l'article [2], la formule de calcul de SSIM entre deux images se fait selon le calcul suivant :

$$SSIM(X, Y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Avec :

- $C_1 = (L * K1)^2$ une constante
- L : la plage de valeurs pour chaque sous pixel (de 0 à 255)
- $K_1 = 0.01$ choisie d'après l'article [2]
- $C_2 = (L * K2)^2$ une constante
- $K_2 = 0.03$ choisie d'après l'article [2]
- μ_X : la moyenne de luminance sur la première matrice
- μ_Y : la moyenne de luminance sur la deuxième matrice
- σ_X : l'écart-type des luminances sur la première matrice
- σ_Y : l'écart-type des luminances sur la deuxième matrice
- σ_{XY} : la valeur calculée qui lie les deux images

4.1.4 étape 2

HexaToBin :

Ce programme prend en entrée une chaîne de caractères correspondant à un nombre hexadécimal, et retourne la chaîne de caractère binaire correspondante. Pour réaliser cette conversion, on utilise d'abord la conversion de la chaîne hexadécimale vers un entier décimal grâce à la fonction decompoHexa définie plus haut. Puis la méthode ".toBinaryString" nous permet de convertir ce nombre décimal en chaîne de caractères binaires.

BinToInt :

Dans le même esprit que la précédente, cette fonction prend en argument un mot binaire sous forme de chaîne de caractères, et en renvoie la conversion décimale sous forme d'entier. Le calcul se fait en sommant les puissances de 2 correspondant au rang des "1" dans le mot d'entrée.

étape2 :

Voici maintenant le gros du programme de l'étape 2. L'objectif de l'étape est de cacher une image dans un support via la méthode LSB en choisissant le nombre de bits que l'on souhaite modifier. Les arguments sont donc les ImageWrapper des deux images support et cachée, ainsi que le nombre de bits souhaités pour chacune des quatre informations contenues dans les pixels (transparence, rouge, vert et bleu).

Nous avons ajouté dès le début de la fonction un test qui limite le nombre de bits demandés à 8, puisque chaque couleur est codée sur 8 bits. Dans le cas où les arguments respectent la condition, on balaye l'image pixel par pixel et on crée pour chaque pixel la chaîne de caractères binaire contenant

les informations. Ces chaînes sont ensuite séparées pour pouvoir différencier le traitement de chaque couleur.

Le traitement se fait grâce à la méthode ".slice" que l'on applique aux chaînes de caractères pour prélever les $8 - n$ bits de poids fort de l'image support et les n bits de poids fort de l'image cachée pour chaque couleur. Comme à l'étape -1, ces 8 bits sont ensuite assemblés et traduits en décimal à l'aide de BinToInt définie au-dessus. On peut ainsi former l'image stéganographiée.

4.1.5 étape 2bis

tailleMaxMessage :

Cette fonction permet de calculer le nombre de bits que l'on peut remplacer selon une marge d'erreur maximale acceptée. Cette erreur est calculée grâce à la fonction MSSIM. Afin de calculer le nombre souhaité, on va effectuer une simulation du pire cas pour chaque bit libéré.

Le pire des cas est lorsque l'image est strictement différente de l'image originale. Ainsi, par définition, pour créer ce cas, il faut changer les bits 1 contre des bits 0 et inversement. Cette image va permettre de donner la borne supérieure de l'erreur lorsque que l'on alloue k bits (k étant un entier entre 1 et 8).

Pour trouver le nombre maximal de bits pouvant être alloué, on crée l'image correspondant au pire cas en incrémentant k d'un pas de 1. Dès lors que l'on trouve une erreur plus élevée que l'erreur maximale autorisée avec l'image du pire cas, alors le nombre de bits que l'on peut utiliser pour cacher de l'information sera de $k-1$.

4.1.6 étape 2ter

MessageCache :

La fonction MessageCache permet de cacher un String dans une image. Elle prend aussi en entrée un entier entre 1 et 8 représentant le nombre de bit que l'on souhaite utiliser pour chaque octet de couleur afin d'y placer les éléments du string à cacher.

Tout d'abord, le message est converti en binaire à l'aide de la fonction TextToBinary. Avant toute modification, on regarde si la place est suffisante par rapport à la place disponible.

D'après [1], lors d'une transmission d'information, il est possible que celle-ci soit perdue. Il est donc recommandé de copier plusieurs fois le message afin d'avoir une plus grande chance de la récupérer dans son intégralité.

Néanmoins, il existe une faille dans cette répétition. En effet, supposons un mot de la forme "abab" de n'importe quelle taille.

Dès lors que l'on va décoder le stégo-médium pour retrouver le message, le programme va retourner une réponse du type "abababababab ...". On remarque donc qu'il est impossible de distinguer à quel moment le message termine.

Pour contrer cette faille, on peut créer un programme permettant de détecter ce type de répétition dans le message. Or, par manque de compétences, nous avons été dans l'incapacité de créer un tel programme.

Afin d'effectuer la répétition du message, on compte le nombre de bits placé dans la matrice. Dès que l'on atteint la taille du message, on revient au début de celui-ci jusqu'à utiliser l'intégralité de la place disponible dans l'image.

Afin de sauvegarder le nombre de bits qui ont été modifiés, on le stocke dans le premier pixel. Cela sera donc utile lors du décodage. En effet, il s'agit en quelque sorte de la "clé".

DecoderMessageCache :

On effectue la fonction "inverse" de MessageCacher : on veut récupérer le message dans l'image.

Tout d'abord, on récupère la place qui a été libérée. On analyse ensuite le premier pixel afin d'avoir le nombre binaire du nombre de bit libéré. Selon ce dernier, on prend l'ensemble des bits dans la matrice. Ces bits vont créer un message binaire qui va être traduit à l'aide de la fonction BinaryToText.

TextToBinary :

Cette fonction va permettre de convertir n'importe quel texte en mot binaire à l'aide du code ASCII.

Tout d'abord, on convertit chaque caractère de la chaîne de caractère en nombre d'après le code ASCII. Comme l'ensemble des caractères disponibles ont pour intervalle [32,255] dans le code ASCII, on ajoute des 0 jusqu'à avoir un mot binaire de 1 octet (ce mot binaire étant initialement un String).

BinaryToText :

Cette fonction permet d'effectuer l'opération inverse de TexteToBinary : on veut transformer le mot binaire en un message de caractère.

On convertit tout d'abord chaque octet (8 bits) en code nombre décimal, puis on convertit ce nombre en caractère à l'aide du code ASCII.

copieMatrice :

Cette fonction permet de copier une matrice de dimension 2 afin d'éviter les problèmes d'adresse lorsque l'on veut modifier la matrice voulue sans modifier son clone. Bien qu'il existe une fonction ".clone()" permettant d'effectuer cette tâche, on observe que celle-ci est limitée à une seule copie.

4.1.7 étape 3

Nous avons conscience que la méthode choisie n'est pas optimale et qu'il y avait beaucoup d'autres possibilités. Cependant nous avons manqué de temps et avons préféré bien coder les autres étapes.

Cryptage_1 :

D'après [1], l'ajout de cryptographie est intéressant dans le cas de la stéganographie puisqu'elle permet d'ajouter une nouvelle couche de sécurité pour cacher encore mieux le message.

Afin d'illustrer ceci, nous avons décidé d'utiliser la méthode de César. La méthode de César consiste à prendre deux alphabets et de le déplacer d'un pas k (voir schéma ci-après).

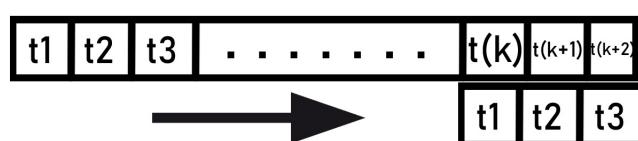


FIGURE 4.6 – Explication cryptage

Le pas k est désigné comme le reste de la division euclidienne de la taille du message par 255. On choisit la taille de l'image car celle-ci reste une information accessible n'importe quand et on choisit 255 car c'est la valeur du code ASCII la plus élevée d'un caractère.

Pour faire le déplacement d'un pas k, on passe par la table ASCII puis on ajoute k à la valeur du caractère en ASCII puis on convertit de nouveau cette valeur en caractère. La chaîne de caractère cryptée peut être utilisée par la fonction MessageCachee.

Decryptage_1 :

On veut décoder le message initialement crypté par la méthode de César. A l'aide du code ASCII et du pas k décrit précédemment, on enlève k à la valeur du caractère en ASCII puis on convertit de nouveau cette valeur en caractère. Ainsi, on effectue l'opération inverse.

4.2 Phases de tests

Afin de simplifier notre phase de tests, nous avons écrit le programme "*test*" de manière à automatiser un peu ceux-ci et rendre la batterie de tests plus rapide à exécuter. Ce programme demande une interaction de l'utilisateur qui doit rentrer le niveau du projet qu'il souhaite tester, et choisir différents paramètres en fonction de l'étape (image à cacher, image à décoder, images à comparer...).

4.2.1 Images carrées de mêmes dimensions en format PNG

Nous avons d'abord testé le programme sur 2 images carrées de mêmes dimensions (256*256 pixels) l'image support 5.1 et l'image cachée 5.2.

On remarque que l'image stéganographiée (5.3) et l'image support (5.1) sont très difficilement discernables à l'oeil nu. Il en est de même pour l'image cachée (5.2) et l'image décodée 5.4).

Lorsque nous appliquons à l'image décodée et son image d'origine la méthode SSIM, on obtient la valeur 0.99718815 qui est très proche de 1. De même, la valeur de EQM est 148.61308, ce qui est faible étant donné qu'il s'agit là d'une erreur quadratique moyenne. Nos observations visuelles sont donc vérifiées par les calculs : la méthode LSB a particulièrement bien fonctionné sur ce test.

4.2.2 Images rectangulaires de mêmes dimensions en format PNG

Pour ce deuxième test, nous avons exécuté le programme sur 2 images rectangulaires de mêmes dimensions (1000 * 600 pixels) : le support est l'image 5.5 et l'image cachée est l'image 5.6.

Nous pouvons tout d'abord voir que l'image stéganographiée (5.7) est moins bien que dans le test précédent puisque l'on peut distinguer l'image cachée en arrière-plan de la courbe. La couleur blanche de l'image support semble empêcher le bon camouflage d'une image colorée comme celle de cet exemple.

Pour ce qui est du décodage de l'image, on garde une image décodée (5.8) de qualité très proche de l'image que l'on avait cachée, l'indice SSIM vaut ici 0.9931076 et l'EQM vaut 251.5646. On en déduit que numériquement, la méthode LSB permet sur cet exemple de garantir un bon décodage de l'image cachée. Cependant, nous avons remarqué que la couleur homogène et claire d'une image support joue un rôle dans la qualité du camouflage d'une autre image aux couleurs plus vives et diverses.

4.2.3 Images en format JPEG

Exécutons maintenant le programme sur 2 images rectangulaires de mêmes dimensions (174 * 290 pixels) mais cette fois en format JPEG : le support 5.9 et l'image cachée 5.10.

On remarque dans l'image stéganographiée (5.11) que les nuances de couleurs sont de moins bonne qualité, des vastes zones "lisses" apparaissent notamment sur le visage de l'homme et en arrière plan. La stéganographie a nettement diminué la qualité visuelle de l'image support. Concernant l'image décodée (5.12), on observe le même phénomène : des zones lisses et des nuances de couleurs violentes.

Si on applique SSIM à l'image décodée et l'image que l'on avait cachée, on obtient 0.99390376, et l'EQM vaut 110.2727, ce qui reste très correct, une fois de plus malgré la perte de qualité visuelle que l'on a noté. Celle-ci est peut-être due à la faible taille des images utilisées (174 * 290), puisqu'une différence sur un pixel sera beaucoup plus marquante pour notre œil que si l'image était de qualité supérieure.

4.2.4 Images de tailles différentes et de format différent

Nous avons ici essayé de cacher une image au format PNG de dimension $600 * 1000$ (5.14) dans une image de dimension $174 * 290$ au format JPEG (5.13).

En cachant une image très grande dans une plus petite, on remarque dans l'image stéganographiée (5.15) que seule la partie en haut à gauche de la grande image correspondant aux dimensions de l'image support est cachée, le reste disparaît. Lors du décodage de l'image (5.16), on obtient sans surprise uniquement la portion de l'image qui a été cachée dans le support.

Il faut aussi noter que la différence de format n'a pas posé de souci dans les opérations menées sur les deux images. De plus, la remarque du test précédent sur les couleurs lisses en avec des nuances grossières est encore vraie ici (cela semble confirmer notre hypothèse de la taille trop petite de l'image).

4.2.5 Fonctionnement des différentes étapes

Étapes -1 et 0 : Le fonctionnement des étapes 0 et -1 a déjà été testé dans les trois premiers tests, puisque nous avons bien caché puis extrait une image dans une autre image de taille identique.

Étape 1 : Afin de tester le fonctionnement de l'étape 1, nous avons exécuté les calculs suivants :

- EQM sur 2 images identiques (2 fois la même image)
- EQM sur 2 images radicalement différentes
- SSIM sur 2 fois la même image
- SSIM sur 2 images radicalement différentes

En théorie, nous attendions un EQM faible sur deux images similaires et très élevé sur deux images différentes, et un SSIM proche de 0 sur deux images différentes et proche de 1 sur deux images similaires.

Les images testées sont les images 5.17 et 5.18

Les résultats obtenus sont sans appel :

- EQM (Image 1 , Image 1) = 0.0
- EQM (Image 1 , Image 2) = 3682.4006
- SSIM (Image 1 , Image 1) = 1.0
- SSIM (Image 1 , Image 2) = 0.4406479

Les résultats correspondent aux attentes, les fonctions répondent donc bien à la problématique de l'étape.

Étape 2 : L'objectif de l'étape 2 est de pouvoir choisir le nombre de bits d'informations sur les couleurs ainsi que la ou les informations de l'image que l'on veut cacher. Par exemple, on peut vouloir cacher 4 bits rouges, 2 verts et aucun bleu d'une image dans une image support.

Utilisons ces deux images les images 5.19 et 5.20

Les résultats suivants sont les obtenus en augmentant petit à petit le nombre de bits changés de la forme suivante : Transparence (T) = n bits changés / Rouge (R) = n bits changés / Vert (V) = n bits changés / Bleu (B) = n bits changés.

- si $n = 0$: image 5.21
- si $n = 1$: image 5.22
- si $n = 2$: image 5.23
- si $n = 3$: image 5.24
- si $n = 4$: image 5.25
- si $n = 5$: image 5.26
- si $n = 6$: image 5.27
- si $n = 7$: image 5.28
- si $n = 8$: image 5.29

En observant les images stéganographiées au fur et à mesure que l'on augmente le nombre de bits remplacés, on remarque que l'image support disparaît peu à peu, laissant apparaître l'image qui était cachée. Le cas où on ne remplace aucun bit (5.21) correspond à l'image support, et le cas où on remplace les 8 bits (5.29) de chaque couleur correspond au remplacement complet du support par

l'image que l'on avait cachée. On peut donc avancer que l'étape 2 de notre code fonctionne puisque l'on choisit les bits à remplacer et leur nombre.

Étapes 2bis, 2ter et 3 : Ces étapes ne sont pas assez avancées ou corrigées pour pouvoir mener des tests du même genre que ceux ci-dessus.

Chapitre 5

Annexe : Images des tests

5.1 Images carrées de mêmes dimensions en format PNG



FIGURE 5.1 – Image support 1

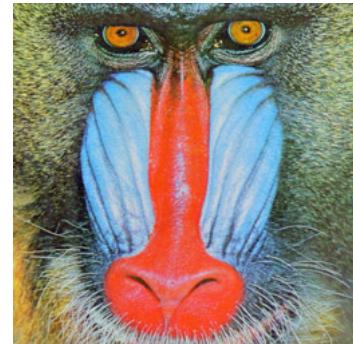


FIGURE 5.2 – Image cachée 1



FIGURE 5.3 – Image stéganographiée 1



FIGURE 5.4 – Image décodée 1

5.2 Images rectangulaires de mêmes dimensions en format PNG

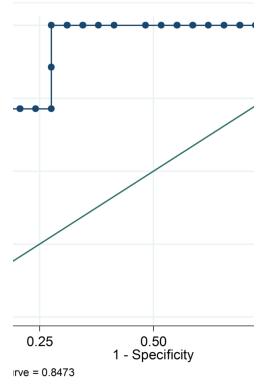


FIGURE 5.5 – Image support 2



FIGURE 5.6 – Image cachée 2

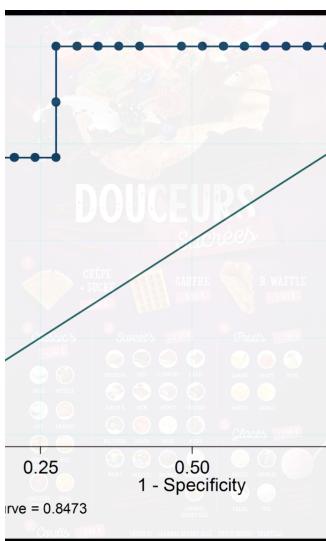


FIGURE 5.7 – Image stéganographiée 2



FIGURE 5.8 – Image décodée 2

5.3 Images en format JPEG



FIGURE 5.9 – Image support 3

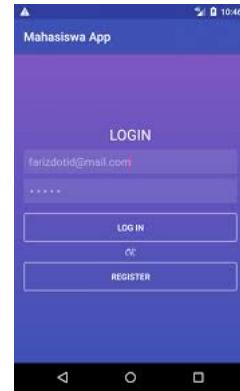


FIGURE 5.10 – Image cachée 3



FIGURE 5.11 – Image stéganographiée 3

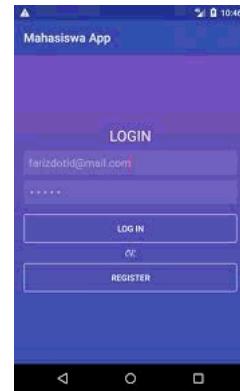


FIGURE 5.12 – Image décodée 3

5.4 Images de tailles différentes et de format différent

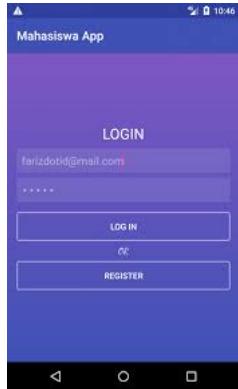


FIGURE 5.13 – Image support 4



FIGURE 5.14 – Image cachée 4

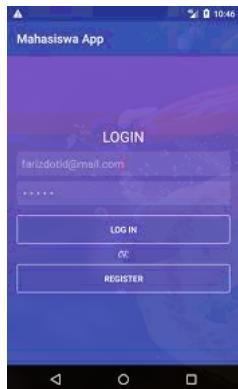


FIGURE 5.15 – Image stéganographiée 4



FIGURE 5.16 – Image décodée 4

5.5 Fonctionnement des différentes étapes



FIGURE 5.17 – Image 1



FIGURE 5.18 – Image 2

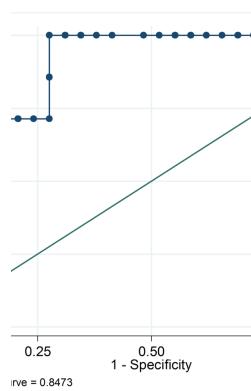


FIGURE 5.19 – Image support étape 2



FIGURE 5.20 – Image cachée étape 2

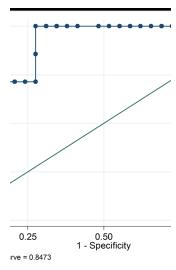


FIGURE 5.21 – LSB : T=0 / R=0 / V=0 / B=0

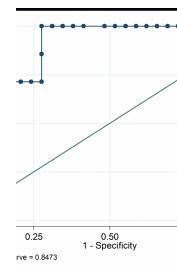


FIGURE 5.22 – LSB : T=1 / R=1 / V=1 / B=1

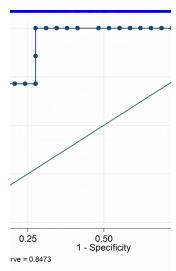


FIGURE 5.23 – LSB : T=2 / R=2 / V=2 / B=2

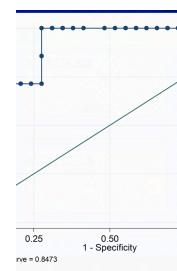


FIGURE 5.24 – LSB : T=3 / R=3 / V=3 / B=3

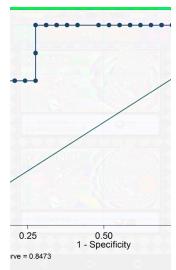


FIGURE 5.25 – LSB : T=4 / R=4 / V=4 / B=4

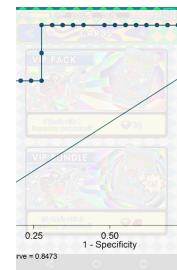


FIGURE 5.26 – LSB : T=5 / R=5 / V=5 / B=5



FIGURE 5.27 – LSB : T=6 / R=6 / V=6 / B=6



FIGURE 5.28 – LSB : T=7 / R=7 / V=7 / B=7



FIGURE 5.29 – LSB : T=8 / R=8 / V=8
/ B=8

Chapitre 6

Bilan du projet

6.1 Bilan global

Ce projet nous a beaucoup appris. Aussi bien au niveau de la programmation qu'au niveau de la gestion de projet. En effet, nous n'avions jamais eu l'occasion de réaliser un tel travail et encore moins en groupe. Les débuts ont été un peu difficiles, nous étions dans un univers de travail peu connu et il fallait donc s'y adapter assez rapidement. De plus, étant le premier projet de l'année, ce dernier nous a quelque peu impressionnés ce qui a entraîné une sorte de flou sur l'ensemble du travail demandé.

Une fois que nous avions trouvé nos aises l'avancée du projet s'est nettement améliorée et nous avons pu progresser au fur et à mesure. Un des points positifs de notre fonctionnement à été une bonne entente ainsi qu'une bonne communication ce qui a permis de pouvoir échanger facilement et de discuter des problèmes rencontrés. Cependant, nous avions parfois mal repartis les tâches en effet nous réfléchissions tous ensemble sur une même étape ce qui nous sûrement fait perdre du temps, essentiellement au début.

Certaines améliorations auraient pu être réalisées sur notre code. Par exemple, lors de l'étape 3 nous aurions pu mettre en place de nouvelles méthodes comme celles présentées dans l'état de l'art mais malheureusement le temps nous à manqué.

Enfin, nous retirons tous une expérience positive de ce projet qui nous servira sûrement pour les années et travaux à venir.

6.2 Bilan individuel

6.2.1 Coralie

Au démarrage de ce projet, je n'avais que très peu de connaissance en codage informatique et cela m'a vite intimidée. En effet, certaines étapes me paraissaient très compliquées à réaliser alors qu'elles étaient finalement plutôt simples une fois que j'avais les bons outils en mains. Bien-sûr, d'autres étapes restent tout de même difficiles à mes yeux mais cela est maintenant beaucoup plus clair.

Ce projet m'a permis d'apprendre à utiliser GitLab de manière fluide mais aussi d'améliorer mes compétences en codage. Nous avons pris le parti de laisser certaines étapes à coder pour Niels car celui-ci avait un niveau un peu plus élevé que le reste de l'équipe. Cependant j'ai pu réaliser certaines fonctions parfois seule ou à l'aide de mes deux autres camarades ce qui a été très enrichissant.

A l'avenir, je pense que nous pourrons aussi nous améliorer sur notre gestion du temps et du travail car celle-ci n'était pas optimale durant le projet. Mais nous avons acquis désormais un savoir qui nous sera très utile.

6.2.2 Hugo

Maintenant ce projet terminé, je dois dire qu'il a été beaucoup plus formateur que ce à quoi je m'attendais. En effet, les étapes du travail étant données et le projet étant étalé sur deux mois, je me suis sans doute laissé penser que le temps était largement suffisant. Or, il n'en a pas été, puisqu'à mon avis, notre répartition des tâches dans le temps et entre les membres du groupe n'était pas optimale et pourrait nettement être améliorée lors d'un projet futur.

À l'issue du projet, je suis maintenant capable d'écrire du code en Scala, langage que je ne connaissais pas ou très peu auparavant. J'ai aussi acquis de solides bases en L^AT_EXet en gestion de projet sur GitLab. Concernant la gestion de projet en tant que telle, j'ai maintenant une idée plus claire de ce qu'il en retourne puisque nous avons cherché à mettre en oeuvre une application concrète du MOOC de Rémi Bachelet.

Il y a encore certains points sur lesquels je peux m'améliorer. Tout d'abord, j'ai pu remarquer un manque de fluidité dans le codage, des erreurs évitables se glissent souvent dans mes programmes, et me font perdre du temps puisqu'il faut ensuite débugger (ce qui est très chronophage, je l'ai découvert). Les problèmes que nous avons rencontrés lors de ces deux mois furent multiples : la difficulté de conjuguer les emplois du temps des différents membres de l'équipe, la gestion en parallèle de ce projet et des révisions de partiels de décembre, ou encore les multiples conflits qui nous ont ralenti sur Git au début du projet. Tous ces moments de "crise" font qu'aujourd'hui, j'ai acquis un certain recul vis-à-vis de la gestion de projet et du développement informatique que je n'avais pas il y a quelques mois et qui me servira pour sûr tout au long de ma carrière.

6.2.3 Niels

Pour mon cas, le projet a été très formateur, aussi bien dans le domaine de la programmation avec Scala qu'en la Gestion de Projet. Personnellement, c'est la première fois que je m'attaque à un projet aussi long en groupe : la gestion de groupe était donc fondamentale. Cela m'a permis de mettre en pratique tout ce que l'on a appris au cours des premiers mois de cours et cela s'est avéré essentiel pour le bon déroulement du projet.

Du point de vue de la programmation, je suis capable d'utiliser le langage Scala initialement vu en cours mais de manière plus restrictive. De même, pour L^AT_EXavec la rédaction du rapport ou encore Git pour l'avancement du Code.

Néanmoins, beaucoup de points sont à améliorer. Pour la gestion du temps, je pense que je devrais demander directement à mes collègues d'avancer sur d'autres fonctions au lieu de rester plusieurs heures sur une fonction sans savoir quoi faire. Bien que la communication était bonne, nous avons eu quelques difficultés pendant les vacances.

6.3 Nombre d'heures par parties du projet

	Hugo BENAMEUR	Coralie DELARBRE	Niels TILCH
<i>Recherches</i>	14	14	14
<i>Codage</i>	32	30	42
<i>Gestion de Projet et Rapport</i>	25	28	14

Chapitre 7

Bibliographie

- 1 : Neil F. Johnson, Sushil Jajodia, *Exploring Steganography : Seeing the Unseen*, 1998, IEEE, pages 26 à 34
- 2 : Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, Ero P. Simoncelli, Image Quality Assessment : From Error Visibility to Structural Similarity, 2004, IEEE, pages 1 à 14
- 3 : <http://www.bibmath.net/crypto/index.php?action=affichequois=stegano/histstegano>
- 4 : <http://www.bibmath.net/crypto/index.php?action=affichequois=stegano/tatouage>
- 5 : <http://www.bibmath.net/crypto/index.php?action=affichequois=stegano/tatouage>
- 6 : <https://www.securiteinfo.com/attaques/divers/steganographie.shtml>
- 7 : <https://securityaffairs.co/wordpress/62191/malware/syncrypt-ransomware.html>
- 8 : <http://www.bibmath.net/crypto/index.php?action=affichequois=stegano/index>
- 9 : https://www.lirmm.fr/icar/presentation/KOUIDER_debut_these.pdf
- 10 : <https://www.techniques-ingenieur.fr/base-documentaire/technologies-de-l-information-th9/cryptographie-authentication-protocoles-de-securite-vpn-42314210/introduction-a-la-steganographie-h5870/>
- 11 : <https://fr.wikipedia.org/wiki/St%C3%A9ganographie>
- 12 : <https://www.01net.com/actualites/les-cyberespions-russes-s-attaquent-aux-diplomates-europeens-par-steganographie-1788751.html>
- 13 : <https://www.01net.com/actualites/comment-des-pirates-cachent-des-mineurs-de-cryptomonnaie-dans-des-fichiers-son-1789969.html>
- 14 : http://pignon.camille.free.fr/save/techniques_stegano.pdf
- 15 : <https://lig-membres.imag.fr/PPerso/membres/donsez/ujf/easrr0203/tatouagestegano/tatouagestegano.pdf>
- 16 : <https://ftp.visionduweb.fr/DOCUMENTATIONS/LINUX/PEN%20TEST/Steganographie%20-%20Techniques.pdf>
- 17 : API ImageWrapper : <http://tncyttop.github.io/top-roadetection/doc/com/tncyt/top/image/ImageWrapper>
- 18 : <https://slideplayer.com/slide/5216018/>