heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Travail de diplôme

Stéganographie
Détection de messages cachés

Professeur responsable : Stephan Robert

Mandant : Gilberto Girardello, Police de Sûreté Vaudoise

Ljupce Nikolov

14 décembre 2008

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Résumé

La stéganographie est l'art de la dissimulation de communication. Son objectif est de pouvoir dissimuler des données qui doivent être tenues secrètes dans un support paraissant anodin. Ces données ne doivent pouvoir être récupérées que par une personne ayant connaissance de l'algorithme utilisé pour la dissimulation, ainsi que de l'éventuel clé saisie.

Ce document a pour but de présenter une application permettant la détection de contenu ayant été stéganographié. Cette application effectue de la stéganalyse, discipline inverse à la stéganographie, dont l'objectif est de pouvoir détecter lorsqu'un support véhicule des données dissimulées.

Le présent document est constitué de quatre parties. La première partie est une description générale de l'application développée, ainsi que des analyses la constituant.

La seconde partie est destinée à l'utilisateur final. Elle comprend des informations concernant la prise en main, ainsi que des possibilités de l'application.

La troisième partie donne des informations plus techniques sur l'application, ainsi que sur les éléments nécessaires lors de son développement. Une explication détaillée des formats pris en charge y figurent notamment.

Enfin, pour finir, la dernière partie de ce document contient les annexes. Principalement constituée du rapport de pré-projet de diplôme ayant précédé le travail actuel, contenant des informations plus détaillées concernant la stéganographie, ainsi que la stéganalyse.

Cahier des charges pour le travail de diplôme

Titre

Détection de messages cachées (Stéganographie).

Professeur responsable

Monsieur Stephan Robert.

Mandant

Monsieur Gilberto Girardello, Police de sûreté Vaudoise.

Résumé du problème

Examen d'un certain nombre de fichiers (JPEG, MPEG, MP3.,...) pour détecter s'ils contiennent un message caché.

Cahier des charges

1. Intégration en tant que script dans le logiciel Encase (il s'agit d'un logiciel forensique utilisé par les forces de police). www.encase.com. Il s'agit dans un premier temps d'apprendre le langage de scripting intégré dans Encase (mélange de Java et C++) et de transposer le logiciel existant avec les corrections pour qu'il fonctionne en tant que script dans le logiciel Encase.
2. Module avec corrections des défauts par rapport à la première version (voir TD « Steganographia » de M. Lifschitz). Les logiciels de manipulation d'images (suppression des yeux rouges, rotation des images, etc.) laissent des « reliques » à la fin des fichiers, diminution des faux positifs.
3. Dans le cadre du script Encase, certaines fonctionnalités du logiciel existant (voir TD « Steganographia » de M. Lifschitz) seront supprimées car le logiciel Encase en possède déjà certaines, entre autres les hash MD5 ainsi que la détection des fichiers correspondants, détection du type de fichier en fonction de sa signature numérique et comparaison avec son extension. Par contre une base MD5 contenant une liste exhaustive de logiciels de stéganographie sera ajoutée (ajout d'un hashset dans la bibliothèque MD5).
4. Nouvelles méthodes/Meilleures détections. Analyse de nouveaux types de fichier PNG, MP3, WMA,WMV, MPEG, MOV,RM. Détection stéganographique avec les bits de poids forts.

Table des matières

Première partie

Documentation Générale

1 Introduction

Ce document décrit une application de détection de contenu stéganographié. La stéganographie regroupe les outils et méthodes permettant de cacher des informations, afin qu'elles soient invisibles pour une entité ne faisant pas partie de la communication. Cette discipline est détaillée dans le pré-projet ayant précédé l'écriture de ce document. Il se trouve en Annexe D.

Ce travail à été proposé par la police de sûreté vaudoise. Dans le cadre de leurs investigations, ils utilisent un produit forensique du nom de EnCase. Ce dernier leur permet la récolte de nombreuses informations concernant l'utilisation faite de la machine examinée. L'objectif de ce travail est d'étendre les fonctionnalités de ce produit, en y ajoutant la détection de contenu stéganographié. Ceci est rendu possible par la présence, de manière native, d'un langage de script. Du nom de EnScript, ce dernier est prévu pour l'automatisation de certaine tâche effectuée sous EnCase. Ces analyses ayant lieu en laboratoire, le temps d'exécution n'est pas un critère déterminant de l'application.

Dans un premier temps, l'application précédente (développée par Daniel Lifschitz en JAVA) doit être portée dans le langage EnScript. Cette étape permettra de pouvoir juger des possibilités offertes par ce langage, et d'évaluer la faisabilité d'une telle application dans ce langage de script.

Si cette première étape se révèle concluante, la seconde étape consistera à étendre les fonctionnalités de l'application, soit par la prise en charge d'autres formats de fichier, soit par la définition de nouvelles méthodes de détection de contenu stéganographique. Le choix des formats, ainsi que des analyses nouvellement développées se basera sur une étude de l'existant, en matière de stéganographie. Les méthodes de détection devront permettre une détection stéganographique efficace. Dans la mesure du possible, elles permettront l'identification du produit utilisé lors de la dissimulation des données.

2 Principe de fonctionnement

L'objectif global de l'application, est de pouvoir parcourir une arborescence de fichiers définis. Pour chacun de ces fichiers, une série d'analyses est effectuée pour permettre de déterminer si ce fichier a pu servir de conteneur stéganographique. Afin d'arriver à cet objectif, plusieurs étapes sont nécessaires, la figure suivante présente un schéma simplifié de ces étapes.

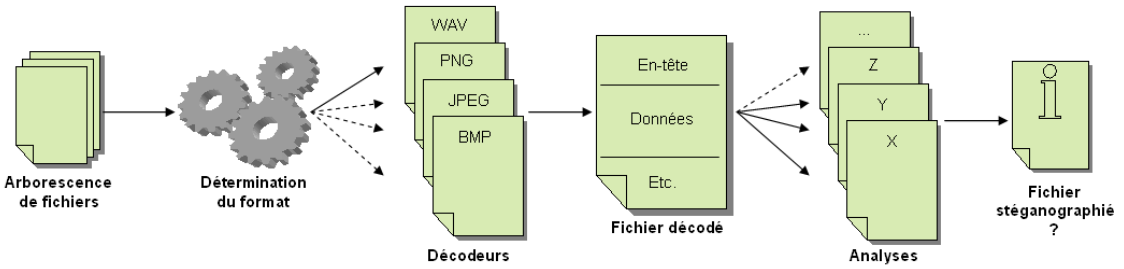

Fig. 1 – Schéma de fonctionnement de l'application.

La première étape consiste à déterminer le format du fichier traité. Cette étape est nécessaire afin de pouvoir déterminer les analyses qui lui sont applicables. Chaque analyse étant uniquement applicable à une palette de formats donnés. Le format peut être établi de plusieurs manières différentes, le plus simple est de se basé sur l'extension du fichier pour déterminer la nature de celui-ci. Mais cette information étant aisément falsifiable, elle ne présent pas un bon indicateur du format. Une seconde manière de procéder est de vérifier la signature du fichier. Chaque format de fichier est reconnaissable par une suite de bytes se trouvant généralement en tout début de fichier. Par exemple, les fichiers BMP commence par les caractères BM codés en ASCII (`0x424D`). L'identification par signature est plus long, mais beaucoup plus efficace. Une modification de la signature rend souvent le fichier inutilisable (ne peut plus être ouvert par les applications usuelles).

En fonction du format, le fichier est passé au décodeur adéquat. Celui-ci, en se basant sur les spécifications du format, segmente le fichier en parties distinctes. Toujours selon le format, une décompression des données est effectuée, sous réserve qu'elles soient compressées. De plus, ce processus permet de vérifier que l'identification du format soit bien correcte. Le décodage n'est possible que si le format du fichier respecte scrupuleusement les spécifications émises. Il en sort un fichier morcelé en plusieurs parties.

Pour terminer, l'étape clé de l'application, l'exécution des analyses stéganographique. En fonction de la nature de l'analyse, certaines parties du fichier sont analysées. Ces analyses donnent une réponse binaire à la question : Ce fichier est-il susceptible de dissimuler du contenu stéganographié ? Le format du fichier définit les analyses qui lui sont applicables.

Une étape supplémentaire n'a pas été mentionnée ici. La raison est qu'elle n'est pas directement liée à l'application. Cette étape consiste à comparer le hash MD5 des fichiers dans l'arborescence à une collection de hash (Hash Set) représentant des fichiers suspects. Ce processus étant fonction intégrante du logiciel EnCase, seul la définition d'un Hash Set est nécessaire. Celui-ci regroupe des fichiers permettant une détection de quelques logiciels de stéganographie.

3 Analyse de marché

Afin de pouvoir guider le choix des développements futurs, une analyse de l'existant s'est rapidement avérée nécessaire. Sur Internet, de nombreux logiciels de stéganographie sont disponibles. On compterait à ce jour plus de 200 produits de stéganographie [19].

Tous ces produits ne jouissent cependant pas de la même popularité. Une analyse de chacun de ces logiciels n'étant clairement pas envisageable, se concentrer sur les plus populaires d'entre eux semble une bonne démarche.

Toujours avec pour objectif de mieux cibler le catalogue de produits existants, l'analyse est étendue au format de fichier ainsi qu'au méthode stéganographique. Cette démarche permet de différentier ce qu'il est possible de faire, et ce qui est vraiment fait.

3.1 Produits

Devant l'absence de réelle étude à ce sujet, une approche spécifique a été définie. Une corrélation entre le nombre de téléchargement sur quelques uns des principaux sites de téléchargement et le nombre de mentions dans des documents traitant de la stéganographie est utilisée.

Il est difficile d'évaluer la fiabilité de cet indicateur. Le téléchargement en soi n'engendre pas l'utilisation du produit. Toutefois, aucune autre méthode n'a pu être trouvée afin de déterminer l'utilisation de ces produits. Des statistiques émanent de l'utilisation du script développé lors de ce travail pourrait servir à en avoir une meilleure vision.

Les sites de téléchargement utilisés comme source sont les suivants :

- http ://www.download.com
- http ://www.telecharger.com (site francophone)
- http ://www.securitysoftwarezone.com (spécialisé dans le domaine de la sécurité)
- http ://www.dodownload.com
- http ://www.freedownloadcenter.com

La grande majorité des produits ressortant de cette analyse sont disponibles pour plate forme Windows. Cependant, afin d'être le plus complet possible, des produits pour Linux et MacOS ont aussi été décrit.

3.1.1 Windows

Comme mentionné précédemment, Windows reste la plateforme la plus représentée. Parmi les logiciels disponibles, plusieurs catégories peuvent être distinguées.

La première de ces catégories correspond à des suites logicielles. Généralement spécialisées dans le domaine de la sécurité, elles proposent dans leurs fonctionnalités la possibilité de stéganographier des documents. On peut signaler Steganos Privacy Suite 2008 et NeoByteSolution Invisible Secrets 4.

La seconde de ces catégories est composée de logiciel de cryptographie. Dans ce cadre là, la stéganographie est simplement vue comme une option supplémentaire. On peut notamment citer LastBit Software Absolute Password Protector.

La dernière catégorie est composée de logiciel de stéganographie pure. WbStego4Open et Hermetic Systems Hermetic Stego 7 sont à placer dans cette catégorie. Pour finir, certaine

société laisse en téléchargement gratuit leurs produits dans des versions ultérieurs. NeoByteSolution a choisi cette alternative et propose Invisible Secrets dans sa version 2.1 en libre téléchargement.

3.1.2 Linux

Sur Linux, les produits de stéganographie sont moins répandus et de plus, sur les sites de téléchargement usuels, Linux est bien moins représenté. Cependant quelques produits ont pu être trouvés.

Parmi eux, il y a notamment Outguess 0.2. C'est un produit découlant du domaine académique assez avancé, il effectue notamment de la compensation statistique qui lui permet d'être difficilement détectable.

WbStego4Open, déjà présent dans la section Windows, est aussi disponible sur Linux. Cela en fait donc un candidat de choix pour une analyse.

StegHide 0.5.1 est lui aussi disponible sur les deux plateformes. Mais il n'est plus mis à jour.

3.1.3 Mac

Tout comme Linux, les produits semblent être moins présents sur MacOS. Pour en signaler quelques uns, on peut mentionner iStego 1.5, Pict Encrypt 2.0 et Cryptix 0.64b.

iStego n'est pas un produit à part entière. Il s'agit d'un frontend utilisant Outguess 0.2 pour la partie fonctionnel. On peut donc utiliser la puissance d'un outils tel Outguess, en y ajoutant la facilité d'utilisation.

Pict Encrypt, quand à lui, est un produit standalone. Il est disponible gratuitement, et semble être le plus utilisé sur MacOS.

Pour finir, Cryptix fait partie des produits de cryptographie ayant comme option de dissimuler les données à l'intérieur des images.

3.2 Format

L'analyse des formats à pour but de cibler les plus importants pour un éventuel développement de décodeur. Se basant sur 20 des logiciels les plus représentatifs actuellement, les formats pris en charge ont été comptabilisés.

Remarque L'analyse ne prend pas en considération les produits permettant de dissimuler des données dans un très large éventail de format de fichier. Il n'apporte rien à la statistique, augmentant simplement le niveau de chaque format (les formats ressortant de cette analyse sont pris en charge par ce genre de produit).

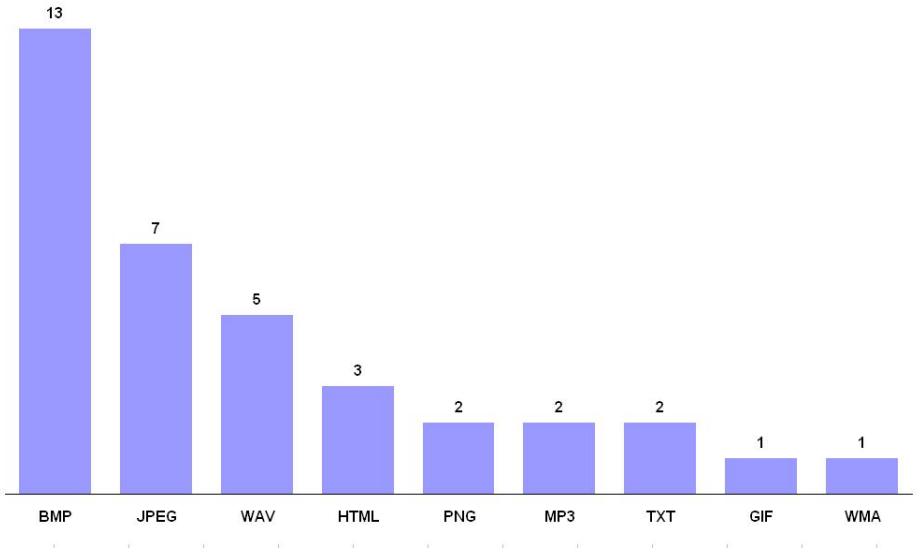

Fig. 2 – Représentation des formats de fichier dans les produits stéganographique.

La figure 2 montre un graphique de la représentation des formats principalement utilisés. Les éléments qui peuvent être tirés de ce graphique sont, d'une part, que le format le plus utilisé est le BMP. Cela provient de la relative simplicité du format qui permet un décodage et une modification de l'image très aisées. Le format JPEG/JFIF arrive second. On note que ces deux formats sont déjà supportés par l'application précédente.

On comprend que les formats d'image restent les plus utilisés. Il n'y a que le format WAV qui soit utilisé dans des proportions significatives. Le graphique fait aussi apparaître des formats tel que le TXT ou l'HTML. Ces formats ne sont cependant pas traité dans ce document.

3.3 Méthodologies de dissimulation

Après avoir traité les produits et les formats privilégiés, il est intéressant de pouvoir déterminer quelles méthodes de dissimulations sont les plus largement utilisées par les produits de stéganographie. Un point très important lors de l'analyse d'un produit est de pouvoir déterminer la méthodologie utilisée lors du processus de dissimulation. Le document [19] fait une synthèse de l'existant. En se basant sur ce document, l'ensemble des méthodologies utilisées actuellement pourrait être classées en 5 catégories :

- Steganographie dans le domaine spatial
- Steganographie dans le domaine des transformées
- Steganographie basée sur le document (Document Based steganography)
- Steganographie basée sur la structure du fichier
- Autres techniques.

3.3.1 Steganographie dans le domaine spatial

La stéganographie dans le domaine spatial regroupe les modifications sur les LSB (Least Significant Bit), ainsi que une autre méthode connue sous le nom de BPCS (Bit Plane

Complexity Segmentation). La méthode des LSB est décrite dans le pré-projet de diplôme ayant précédé ce travail (voir Annexe D, section 4 en page 70).

Le méthode du BPCS [20] a été proposée avec comme objectif de fournir un grande capacité de dissimulation. Elle sectionne l'image en petits blocs, qui sont ensuite définis soit comme porteur d'information, soit comme bloc de bruit. La dissimulation se base sur le fait que l'oeil humain ne discerne pas bien les modification intervenues dans des zones fortement bruitées (ou complexes). Ces dernières peuvent être modifiées sans que la perception de l'image ne change. La grande capacité est apportée par la modification possible dans tout les plans de bit. Contrairement à la méthode du LSB, qui est aussi applicable au fichier audio, Le BPCS n'est utilisable que sur des images.

Malgré ses apparentes qualités, Le BPCS n'est pratiquement pas utilisé dans les produits actuels. Il lui est souvent préféré la modification LSB, de par son implémentation très facile. Qtech-HV est le seul produit ayant été trouvé qui utilise cette technique. Il s'agit d'un logiciel développé à l'issue d'une étude menée sur le BPCS.

Dans le domaine spatial, le LSB reste largement la méthode la plus utilisée. Des produits tels que Invisible Secret, WbStego4Open et Hermetic Stego sont autant de logiciels utilisant cette méthode dans des implémentations très différentes.

3.3.2 Steganographie dans le domaine des transformées

erLa stéganographie dans le domaine des transformées est majoritairement utilisée dans le cadre de dissimulations effectuées sur le format JPEG/JFIF. En lieu et place d'utiliser les pixels de l'image pour caché l'information, ce sont les coefficient DCT de la transformée en Cosinus Discret qui sont utilisés.

Comme dans le domaine spatial, la méthode principalement utilisée est celle des modifications de LSB, mais les modifications sont, cette fois, appliquées au coefficients DCT et non plus directement aux valeurs des pixels.

L'énorme avantage de ce type de méthode est la large prolifération du format utilisé. En effet, les images au format JPEG/JFIF sont de loin les fichiers le plus présents sur le World Wide Web. Tous les jours, des millions d'images JPEG/JFIF sont échangées via e-mail. Un échange de ce type est anodin.

La stéganographie dans le domaine des transformées sur support JPEG semble être une solution de choix. Toutefois, la complexité du format JPEG fait que très peu de logiciels dissimulent réellement les données de cette manière. La plupart des implémentations se contentent d'ajouter les données en fin de fichier.

Jsteg fut le premier produit à implémenter cette méthode. Les produits découlant du domaine académique, tels que F5 ou Outguess, suivent la même voie. Un seul produit commercial semble agir dans le domaine des transformées, il s'agit de Steganos Privacy (anciennement Security) Suite.

3.3.3 Steganographie basée sur le document

La stéganographie basée sur le document est utilisée pour les formats textuels. La dissimulation se fait à l'aide d'ajout de caractère tel que des espaces ou/et des tabulations en fin de ligne. Cela est utilisé pour coder l'information cachée. Les formats tel que l'HTML ou les fichier TXT sont utilisés comme conteneur. Snow est un exemple de produit implémentant cette méthode.

3.3.4 Steganographie basée sur la structure du fichier

La stéganographie basée sur la structure du fichier se sert des espaces non utilisés pour la dissimulation de contenu. Cela est rendu possible par le fait que les décodeurs ne lisent pas les données contenues à ces endroits. Pour la plupart des formats, la dissimulation est effectuée en fin de fichier. Les spécifications de format définissent généralement une suite de bits indiquant la fin du fichier. Similairement, un champ dans l'en-tête du fichier spécifie la taille des données. Un produit utilisant cette méthode se contentera d'appondre les données relatives au contenu caché au fichier servant de conteneur.

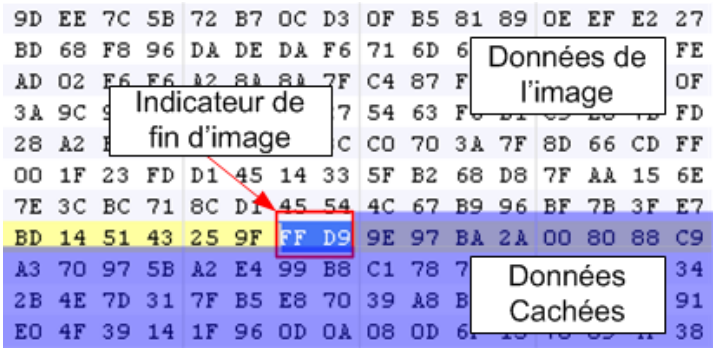

FIG. 3 – Exemple d'ajout en fin de fichier (JPEG/JFIF).

Une autre approche est d'utiliser des champs spécifiés dans la norme du format. En prenant exemple sur les formats JPEG/JFIF ou PNG, ils définissent des champs permettant de saisir des commentaires. Ces champs peuvent être détournés de leur usage afin d'y dissimuler des données. Les décodeurs ne tiennent pas compte du contenu de ces derniers, la modification est donc invisible. Invisible Secret 2.1 utilise les champs de commentaire du format JPEG/JFIF pour camoufler l'information.

L'avantage de cette méthode est qu'elle ne limite pas la taille des données cachées. Cependant, elle modifie la taille du fichier dans sa globalité. Dès lors, en fonction de la taille du fichier caché, le bon conteneur devra être choisi pour ne pas éveiller de soupçons (un fichier PDF de 1 Mb « dissimulé » dans un fichier JPEG risque d'être suspect).

Le niveau de sécurité de ce type de stéganographie peut être vu comme très faible. La détection est très aisée. Si aucun chiffrement n'a été entrepris sur les données, un simple éditeur hexadécimal peut être utilisé pour les récupérer.

Remarque Certains logiciels sont faussement placés dans cette catégorie. Ils possèdent un fonctionnement proche, ajout de données en fin de fichier. Toutefois, cette opération est effectuée de manière aveugle. Il n'est donc plus question de stéganographie se basant sur la structure du fichier. Hiderman et Data Stash fonctionnent de cette manière.

Les méthodologies de dissimulation ne sont cependant pas figées aux seules catégories énoncées ici, de nombreuses autres existent, mais ne sont pas vraiment représentées dans les logiciels disponibles actuellement. Rien que dans les catégories énoncées plus haut, la représentation des produits n'est pas vraiment homogène. Les modifications dans le domaine

spatial, ainsi que la stéganographie basée sur la structure de fichier sont les catégories qui semblent être les plus représentatives de l'offre logiciel actuelle.

Dans le domaine spatial, seul la modification LSB est réellement utilisée. Le BPCS semble à priori prometteur, mais n'est pas utilisé. Il semble que la plupart des acteurs de ce marché souhaitent proposé un logiciel sans vouloir investir beaucoup de temps à la conception de ce dernier. La modification LSB effectuée dans le domaine spatial est très facile à implémenter, contrairement au BPCS qui requiert plus de temps.

Le constat est le même hors du domaine spatial. Si l'on considère les applications effectuées au format JPEG, force est de constater que les applications agissant dans le domaine des transformées ne sont pas très courant. Une fois de plus, l'écriture d'un tel logiciel requiert le développement (ou la modification) d'un encodeur/décodeur JPEG, tâche qui ce révèle très ardue. Alors que le simple ajout de données en fin de fichier ne requiert aucun développement préalable. Mise à part le format BMP, les implémentations existantes dans d'autres formats usent majoritairement de la structure du fichier et non pas des données pour dissimuler un message. Quand cela ne se résume pas simplement à appondre les données à la fin d'un fichier, peu importe sa structure.

En résumé, mise à part les produits venant du monde académique tels que F5, Outguess ou encore Qtech-HV, il semble que peu de sociétés investissent du temps et de l'argent dans le domaine de la stéganographie.

4 Description des analyses

Cette section décrit les analyses implémentées dans le script développé. Certains logiciels cités en section 3.1 n'ont pas pu faire l'objet d'une analyse. Plusieurs raisons peuvent être mentionnées pour expliquer cela. En effet, la plupart des logiciels sont payant et la version de démonstration téléchargée est trop restrictive (Taille des fichiers dissimulé limité, etc..). Ce manque de liberté ne permet pas d'analyser le fonctionnement du logiciel convenablement.

Pour chacunes des analyses, une probabilité théorique de faux positifs est estimée. Elle est accompagnée des résultats observés lorsque l'analyse est effectuée sur une base de test. Cette base est composée de 300 images Bitmap, 100 images JPEG, 20 fichier PNG ainsi que WAV. Il a aussi été ajouté 60 fichiers de tout type (txt, pdf, exe, etc...). Les proportions de fichiers sont en rapport avec le nombre d'analyses implémentées pour chacun des formats.

Deux catégories d'analyses sont à distinguer : Les analyses statistiques et les analyses de signatures. Pour la première d'entre elles, certaines caractéristiques du fichier sont utilisées pour déterminer si le fichier est susceptible de dissimuler des données. Ces méthodes ne sont généralement pas liées à la détection d'un logiciel particulier, mais d'une famille de produits utilisant la même méthodologie (LSB, etc...). De ce fait, l'analyse positive informe que le fichier en question a une certaine probabilité d'être conteneur stéganographique. Aucun indice supplémentaire n'est donné permettant la mise au point d'une méthode de récupération de ces données. De plus, se basant sur les caractéristiques d'un fichier, ces analyses sont souvent limitées à un type de fichier particulier. Dans le cadre des images, de nombreuses méthodes sont uniquement adaptées à l'analyse d'images naturelles (paysages, portrait, etc..). Les images dites synthétiques (plan, dessin, etc..) ne présentent pas les mêmes spécificités, les résultats sont donc aléatoires.

La deuxième de ces catégories se base sur certains artefacts laissés lors de la dissimulation par les logiciels. Ceux-ci peuvent être de nature intentionnels, ou liés au processus de

recouvrement des données. Dans le premier cas, cela peut être l'ajout du nom du produit ou de l'auteur dans les données. Dans le second cas, les informations permettent au logiciel de connaître la manière de récupérer les données au sein du fichier. Les informations utilisée peuvent être la taille des données dissimulées, un indicateur de fin de fichier, ou encore une signature indiquant que les données sont chiffrées. Par l'analyse du comportement du logiciel selon différents cas de figure, des tests peuvent être définis. Ceux-ci autorisent la détection d'un contenu stéganographié à l'aide d'un logiciel spécifique, pour lequel l'analyse a été écrite. L'analyse par signature permet d'avoir des résultats plus précis que par analyse statistique. Si l'algorithme utilisé ne respecte pas le principe de Kerckhoffs (adapté à la stéganographie), alors la simple connaissance du produit utilisé permet de récupérer les données cachées.

Dans le cadre de ce travail, ce sont principalement des analyses de signature que ont été développées. Elles permettent une détection plus fiable et donne un supplément d'information non négligeable aux personnes investigant sur le sujet. Les analyses statistiques présentes offrent un supplément de preuve permettant d'identifier un fichier jugé suspect.

4.1 Analyses de consistance

Le but de ces analyses est de vérifier que la structure du fichier analysé respecte les spécifications du format. Chaque format offre différents endroits où ajouter des données. Ces endroits sont tout simplement ignorés par la majorité des décodeurs. Il s'agit d'une méthode très simple de dissimulation de données utilisée par la stéganographie basée sur la structure du fichier (voir 3.3.4 en page 7). Ces analyses sont parties intégrantes des décodeurs de fichiers.

4.1.1 Consistance des fichiers BMP

Le format BMP offre deux zones permettant de cacher des données :

1. En fin de fichier.
2. Entre l'en-tête et les données d'image.

Après avoir lu le nombre de ligne est de colonne étant mentionné dans l'en-tête, la lecture du fichier est interrompue. Toutes données se trouvant après le dernier pixel de la dernière ligne ne sera pas traité.

Le second emplacement tire parti du faite que la taille de l'en-tête ainsi que la position du début de l'image ne sont pas fixes. Deux champs dans l'en-tête[1] permettent de définir ces valeurs. En jouant sur l'un de ces deux paramètres, il est possible de libérer de la place au sein du fichier. Cette espace peut être utilisé pour y placer des données.

4.1.2 Consistance des fichiers JPEG

Deux emplacements sont offerts par le format JPEG pour la dissimulation de contenu :

1. En fin de fichier.

[1] Le BMP spécifie deux en-têtes : Le File Header et le DIB Header. La taille de l'entête se trouve dans le DIB Header est spécifie la taille de celui-ci. Le début de l'image est spécifié dans le File Header.

2. En tant que commentaire.

La spécification JPEG définit un indicateur de fin de fichier donné par `0xFFD9` (voir figure 3 en page 7). Le décodage du fichier s'arrête dès la lecture de cette valeur sur 16 bits.

Le format JPEG offre la possibilité d'ajouter des commentaires aux images. Ils ne sont ni limités en nombre ni en taille. Cette seconde option peut être utilisée pour dissimuler des données.

4.1.3 Consistance des fichiers PNG

Le format PNG est proche du format JPEG au niveau des possibilités offertes en matière d'espace de dissimulation.

Un premier emplacement est en fin de fichier. Similairement au JPEG, le PNG définit un indicateur de fin de fichier. Il est représenté par la chaîne de caractères **IEND** en ASCII. Comme spécifié en section 18 (page 49), cet indicateur est suivi de 4 bytes de CRC.

Le second emplacement correspond à dissimuler dans les zones de commentaires. La norme PNG en définit plusieurs types ; Un descriptif de chacun d'eux est donné en section 18 (page 49). La quantité de données contenue dans ces zones est additionnée et comparée au seuil défini.

4.1.4 Consistance des fichiers WAV

Le format WAV, tout comme le format PNG, est formaté en blocs (*Chunks*) . Les spécifications du format définissent plusieurs type de chunks pouvant à priori être utilisés pour y dissimuler des données. Ce sont principalement des champs textuels. Cela pose problème, car l'indicateur de fin de données utilisé est le caractère null (`0x00`). Si les données dissimulées ne se limitent pas à des données textuelles, le décodage de ces blocs (si toutefois ils ne sont pas simplement sautés) sera problématique.

Le seul emplacement pouvant être, de manière sûr, utilisé pour stocker des données est la fin du fichier. Chaque bloc possède un champ spécifiant la longueur des données contenues (voir 19 en page 52). Les données peuvent être appondues après le blocs « Data », ce dernier étant le dernier bloc normalement contenu dans le fichier.

4.1.5 Probabilité de faux positifs

Il est difficile de borner la probabilité de faux positifs. Des commentaires dans un fichier JPEG ne sont pas forcément des informations qui ont voulues être dissimulées. Il peut s'agir de données anodines. Lors de certaines opérations, les applications de traitement d'images laissent des artefacts à la fin des fichiers. Cela est, par exemple, observé sur les fichiers BMP. Une simple opération de rotation de l'image engendre la détection de 1 Kb de données dissimulées en fin de fichier. Dans le cas de photo prise par un appareil numérique, certains modèles ajoutent des données après le marqueur de fin JPEG.

Même si cette probabilité ne peut pas être bornée, elle est relativement élevée. Une détection positive ne doit pas être prise comme une preuve mais elle donne cependant une indication aux enquêteurs qui permette de restreindre leur domaine d'investigation.

4.2 Analyse RS

Le concept de l'analyse RS est détaillé dans le pré-projet de diplôme ayant précédé l'écriture de ce document. Il se trouve en Annexe D section 6.2 en page 82.

4.2.1 Probabilité de faux positifs

La probabilité de faux positifs pour cette analyse statistique est difficile à quantifier. Théoriquement, se basant sur le document [17], cette probabilité est faible. Cette analyse a été définie en premier lieu pour les images naturelles (paysages, animaux, etc...). La détection pour des images synthétiques (graphiques, plans, etc..) est plus aléatoire. L'analyse se basant sur la quantification de bruit sur un groupe de pixels, les images fortement bruitées peuvent engendrer une détection positive.

Dans la pratique, la probabilité de faux positifs observée est relativement élevée. La figure 1 relève le pourcentage de faux positifs en fonction du seuil défini pour la longueur minimal du message. Cette longueur est définie à l'aide de l'analyse RS.

Paramètres messageLength	Détections	Faux positifs	% Faux positifs
0.125	54	27	50
0.15	46	21	45.7
0.175	35	14	40
0.2	33	13	39.4

TAB. 1 – Détections et faux positifs en fonction du seuil de longueur de message minimale.

De manière très naturelle, plus le seuil spécifié est élevé, plus la probabilité de faux positifs baisse. Cela se fait au détriment de l'efficacité de détection. La valeur reste très élevée. Cependant, ces résultats sont à relativiser. L'analyse se basant sur la rugosité d'une groupe de pixel, l'ajout intentionnel de bruit uniforme à une image risque de résulter à la détection de cette image comme porteuse d'un message. Sur les 27 faux positifs, 10 sont dues à l'application de bruit à l'image.

Ces faux positifs peuvent être facilement détecté par les investigateurs par simple analyse visuelle. De plus, il n'est pas très courant d'appliquer ce genre de filtre à l'image. Et ce nombre peut pratiquement être divisé par deux. Dans la base de référence, chaque image est présente à deux reprises : la version normale et une version avec rotation à 90°.

Malgré tout, l'analyse RS ne peut pas être utilisée en tant qu'analyse unique. Elle sert principalement à venir confirmer les résultats d'une autre analyse. L'analyse viendra diminuer la probabilité de faux positifs de la seconde.

4.3 InThePicture

InThePicture dans sa version 2.2, est un shareware vendu 25$. Le produit ne semble toutefois plus être maintenu. La partie traitant du produit sur le site de l'éditeur n'est plus disponible. Ce dernier s'est converti au développement d'application sur iPhone.

Seul le format BMP 24 bits est supporté par l'application. La dissimulation a lieu dans le domaine spatial par modification LSB. Afin d'augmenter la capacité de dissimulation, InThePicture modifie les 2 bits de poids les plus faibles. Le processus commence au début des données de l'image et est effectué de manière séquentiel.

Avant de dissimuler les données, l'application inscrit une signature intentionnelle. Cette dernière vaut "ITP !" (Initiales du produit). Cette signature ne peut toutefois pas être lue directement au sein du fichier. Tout d'abord, l'écriture des bytes n'est pas effectuée de manière linéaire. Le codage des bytes entrepris par InThePicture est le suivant :

1. `12 34 78 56`

Il est ensuite nécessaire de déchiffrer les données. InThePicture chiffre les données en utilisant deux clés fixe. Elles ont été crackée par Guillermito[16] et sont données sous forme hexadécimale :

1. `36 7B 4C F5 4C EA 07 02 06 33 2E 18 36 0E 08 06`
2. `C5 02 CE 61 00 ED`

Chacune de ces deux clés sont utilisées pour une opération spécifique sur les données. La première de 16 bits est utilisée pour une opération de soustraction. La deuxième, de 6 bits, est utilisée pour une opération logique XOR. Le déchiffrement est effectué en soustrayant tout d'abord la valeur de la première clé, puis en faisant un XOR avec la seconde. Les clés sont réutilisées sur toute la longueur des données.

4.3.1 Probabilité de faux positifs

La probabilité de faux positifs est inversement proportionnelle à la taille de la signature. Dans le cas de InThePicture, la signature est de 32 bits. Il y a donc 1 chance sur 2^{32} de retrouver cette même valeur. La probabilité de faux positifs est donc faible.

L'efficacité de la détection est confirmée en pratique. Aucun fichier n'est détecté comme faux positifs et l'ensemble des fichiers ayant été modifié par InThePicture sont correctement détectés.

4.4 InPlainView

InPlainView est similaire à InThePicture dans le sens ou il utilise aussi la dissimulation par modification des LSB dans le domaine spatial. Seul les fichiers BMP 24 bits sont pris en charge. Contrairement à la précédente application, seul le bit de poids le plus faible est modifié. La dissimulation est effectuée de manière séquentiel en commençant par le début du fichier.

L'application laisse une signature de 5 bytes en début de fichier. Cette signature est formée de plusieurs parties :

– 2 bytes à 0
– 2 bytes contenant la taille des données dissimulées
– 1 byte définissant si un mot de passe a été défini (0 aucun mot de passe, 1 un mot de passe est défini)

Contrairement aux données qui peuvent être chiffrées, cette signature est ajoutée en clair. Comparativement à InThePicture, la signature apposée n'est pas unique, elle est fonction de la taille des données dissimulées. Il est cependant possible de borner cette valeur. La dissimulation prenant place dans l'unique bit de poids le plus faible, les valeurs possibles sont contenues dans l'intervalle suivant.

$$\left]\, 0,\ \frac{3*h*l}{8} - 5\right]$$

h et l étant la hauteur et largeur de l'image. La soustraction correspond à la longueur de la signature. Le 0 n'est pas compris dans l'intervalle car l'insertion d'un message de taille nulle n'est pas très utile.

4.4.1 Probabilité de faux positifs

Contrairement à InThePicture, la probabilité de faux positifs n'est pas statique. Cette valeur est dépendante de la taille de l'image. Plus l'image est grande, plus cette probabilité sera élevée. Elle peut être donnée par la formule suivante.

$$p = \frac{(\frac{3*h*l}{8} - 5)*2}{2^{40}}$$

Le numérateur correspond aux valeurs pouvant être prise par les trois derniers bytes. Pour une image de 1280x1024, cette probabilité vaudra $8.94 * 10^{-7}$. Cette probabilité reste donc faible.

La vérification expérimentale de ce résultat ne confirme pas ce chiffre. Sur la base de référence, deux fichiers ont faussement été identifiés comme conteneur InPlainView.

4.5 JSteg

JSteg et son successeur JStegShell dissimulent les données en modifiant les coefficients DCT de l'image JPEG. Ils font partie des rares logiciels se basant réellement sur le format JPEG pour la dissimulation de données. JSteg se base sur une librairie existante[2], à laquelle il ajoute une fonction permettant de stéganographier un message.

Les modifications sont effectuées en parcourant les coefficients DCT de manière séquentielle. Les modifications sont appliquées aux coefficients DCT quantifiés. Ceux dont la valeur est égale à 0 ou 1 ne sont pas modifiés. Lors de la déquantification du coefficient, ce dernier est multiplié par la valeur se trouvant dans la table de quantification. Dès lors, le passage d'une valeur 0 à 1, ou inversement, engendre de grosses perturbations. Le fichier JPEG résultant risque de s'en trouver altéré de manière notable.

Le processus d'insertions commence au début des données d'images compressées. Trois champs distincts peuvent être définis :

1. 5 bits spécifiant la taille (en bits) du prochain champ.
2. La taille des données dissimulées (en byte).
3. Les données.

[2]Independent JPEG Group's Library (IJG). http ://www.ijg.org.

Par rapport à ce qui est défini au dessus, JStegShell ajoute une signature à la fin des données. Cette dernière correspond à la chaîne de caractère « korejwa », du nom de l'auteur du logiciel. Cette dernière n'est pas présente dans tous les cas. Le programme ne vérifie pas la taille des données dissimulées. Si elles excèdent la capacité du conteneur, elles sont simplement tronquées et par la même occasion la signature précitée.

Le peu d'éléments ajoutés par Jsteg (dans sa version originale) ne permettent pas de constituer une signature de ce produit. Seul JStegShell, sous condition que sa signature spécifique n'ait pas été tronquée, est détectable.

4.5.1 Probabilité de faux positifs

La détection se limitant à JStegShell, la présence d'une signature de 7 bytes permet d'avoir une probabilité de faux positifs faibles. En ne tenant compte que de cette signature, cette probabilité est de l'ordre de $\frac{1}{2^{56}}$. Ce chiffre s'amoindrit encore du fait de la présence du paramètre spécifiant la taille des données.

La limitation au seul JStegShell, ainsi que la possibilité que la signature soit tronquée diminue l'efficacité de cette analyse.

4.6 Invisible Secrets 2.1

La version 2.1 n'est pas la dernière version du logiciel Invisible Secrets. Cependant, son éditeur a décidé de laisser cette ancienne version en libre téléchargement, comparativement aux versions plus à jour coûtant environ 40$. Cela la rend plus accessible à un utilisateur lambda.

Le produit possède une interface graphique de type "wizard" permettant de guider l'utilisateur au travers des différentes étapes. Le produit gère les conteneurs de type BMP et JPEG. Il est possible de compresser les données ainsi que de les chiffrer (utilisation de l'algorithme Blowfish) avant la dissimulation.

Le produit a la bonne idée d'informer l'utilisateur sur la technique de dissimulation employée pour chacun des format de fichier géré (figure 4). Pour le format JPEG, les informations sont dissimulées dans les zones de commentaires. Quant au BMP, la dissimulation est effectuée par modification des LSB.

Fig. 4 – Informations données par Invisible Secrets. A gauche, pour le format BMP. A droite, pour le JPEG.

L'analyse de ce produit se focalise sur le format BMP. Le descriptif fourni n'est pas assez détaillé concernant la méthode utilisée. La modification pouvant être effectuée de biens des manières, « LSB replacement » ou « LSB matching », séquentielle ou suivant un ordre pseudo-aléatoire, avec ou sans compensations statistiques.

LSB Replacement vs LSB Matching Le "LSB Replacement" consiste à surécrire les bits de poids faible en se basant sur les bits des données à dissimuler. Le "LSB Matching" incrémente ou décrémente la valeur du **byte** de manière aléatoire. Cela permet d'éviter la formation de paires de valeurs (Pairs of values) inhérentes à la méthode de surécriture. La lecture des données dissimulées reste cependant la même.

4.6.1 Analyse statistique du produit

L'analyse des fichiers modifiés à l'aide du produit, en utilisant un éditeur hexadécimal, permet de mettre en évidence certaines particularités des opérations effectuées sur ce dernier. La première chose surprenante est que, même pour un fichier de très petite taille, une grande partie des bytes de données d'images ont été modifiés. En s'aidant d'un filtre permettant de mettre en évidence le dernier plan de bits (le plan LSB, figure 5), on remarque que ces LSB ont majoritairement la même valeur. Cela dépendant de la taille du fichier dissimulé, plus le fichier est petit, plus cette proportion devient grande.

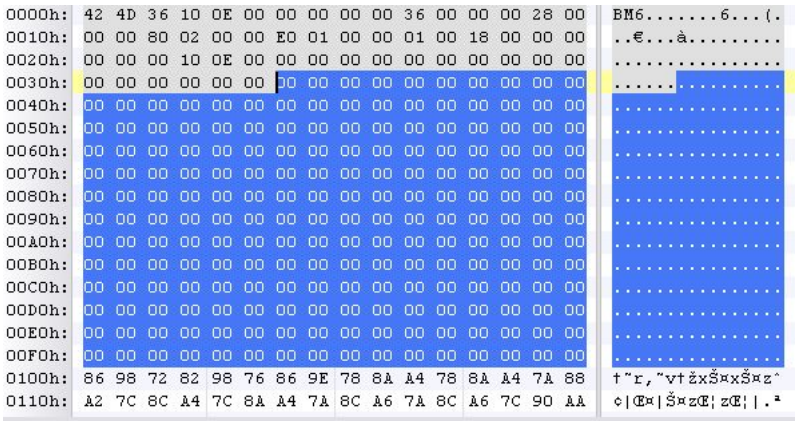

FIG. 5 – Vue LSB du début d'un fichier modifié avec Invisible Secrets 2.1.

De par ces observations, il est déduit un trait de fonctionnement particulier de ce produit. Celui-ci, en plus de camoufler les données au sein des LSB de l'image, modifie l'entier des LSB n'ayant pas été impacté par la dissimulation à la même valeur. Cette caractéristique engendre une modification de certaines statistiques liées à l'image. En considérant l'histogramme du nombres d'occurrences de chaque valeur de byte, la modification des LSB aura comme conséquence la création d'une grande asymétrie entre les bytes de valeur paire et impaire (figure 6).

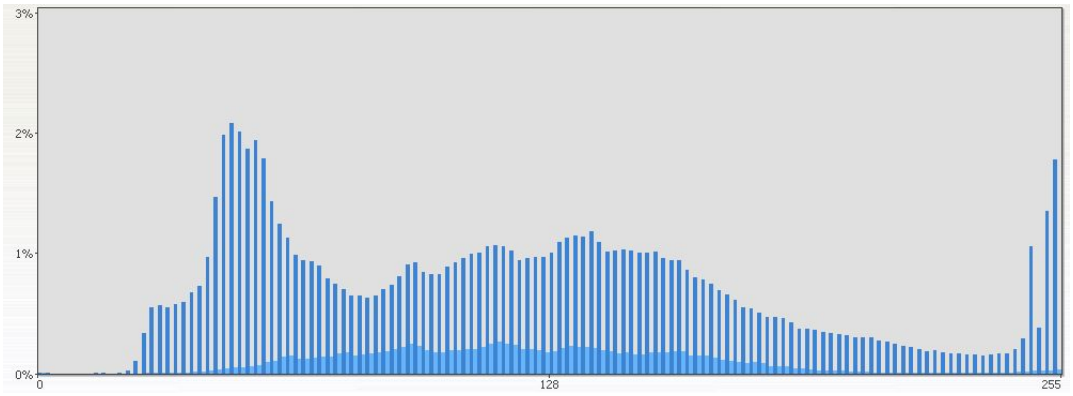

FIG. 6 – Histogramme des valeurs de byte d'une image traitée avec Invisible Secrets 2.1.

En partant de l'hypothèse que le plan de bit LSB pour une image naturelle est uniformément distribué (figure 7), l'asymétrie entre les indices paires et impaires peut être utilisée afin de déceler une altération de l'image due à l'utilisation d'Invisible Secrets. Cette hypothèse peut être vérifiée de manière empirique.

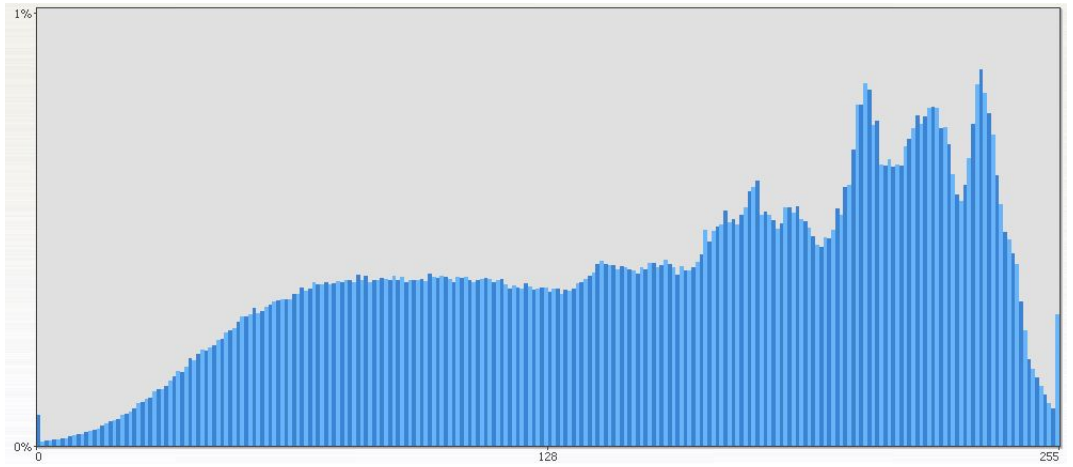

FIG. 7 – Histogramme de couleurs d'une image naturelle.

Se basant sur un échantillon de 10 images, l'analyse de la distribution des bytes permet de confirmer l'hypothèse décrite plus haut. Le tableau de distribution (tableau 2) des valeurs pairs et impairs en confirme les résultats. Parmi les dix images, deux ne sont pas naturelles. Elles sont facilement reconnaissables car elles ne vérifient pas l'hypothèse d'uniformité de distribution des LSB (5.bmp et 1.bmp). Ce sont des images synthétiques qui sont caractérisées par des transitions de couleurs abruptes, ainsi qu'un faible nombre de couleurs. L'histogramme de valeurs de byte montre clairement cette caractéristique (figure 8).

Nom image	Pairs (%)	Impairs (%)
plage_clean.bmp	49.9	50.1
neige_clean.bmp	48.7	51.3
foret_clean.bmp	50.5	49.5
5.bmp	71.7	28.3
890.bmp	49.8	50.2
1.bmp	27.2	72.8
gaula.bmp	51.1	48.9
SF10.bmp	50.4	49.6
spa.bmp	50.5	49.5
raccoons_clean.bmp	50	50

TAB. 2 – Distribution des LSB sur images non stéganographiées.

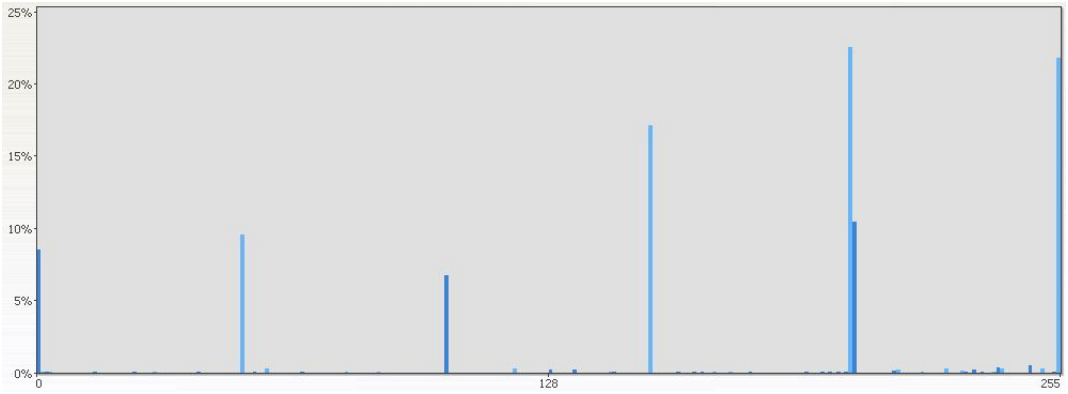

FIG. 8 – Histogramme des couleurs d'une image synthétique.

Maintenant que l'uniformité de distribution à été démontrée sur le LSB au sein des images BMP naturelles, une analyse de l'écart qu'induit la dissimulation à l'aide du logiciel Invisible Secrets 2.1 est effectuée. Cela permet de quantifier cet écart pour plusieurs tailles de fichier caché. Trois tailles de fichier sont utilisées : 1 Byte, 50 KBytes, 100 KBytes. Le conteneur utilisé étant une image de 640x480, le fichier de 100 Kbytes permet de s'approcher de la capacité maximum du conteneur, alors que le fichier de 1 Byte fait l'inverse. Pour commencer, les fichiers ne sont pas chiffrés, ni compressés.

Les figures 9, 10 et 11 montrent les histogrammes de l'image après dissimulation.

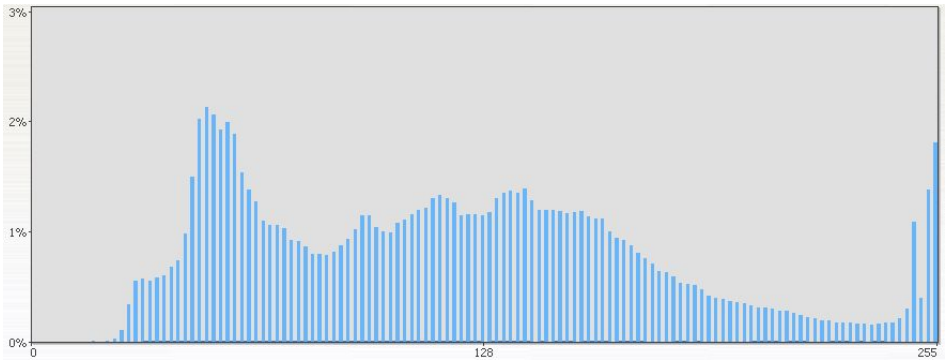

FIG. 9 – Histogramme fichier 1 Bytes.

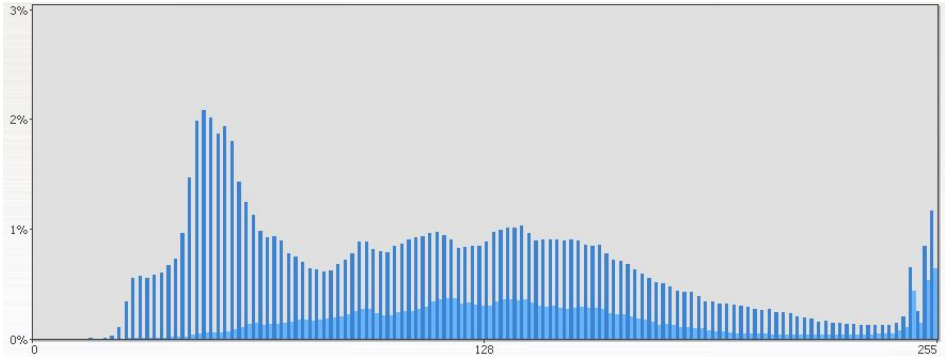

FIG. 10 – Histogramme fichier 50 KBytes.

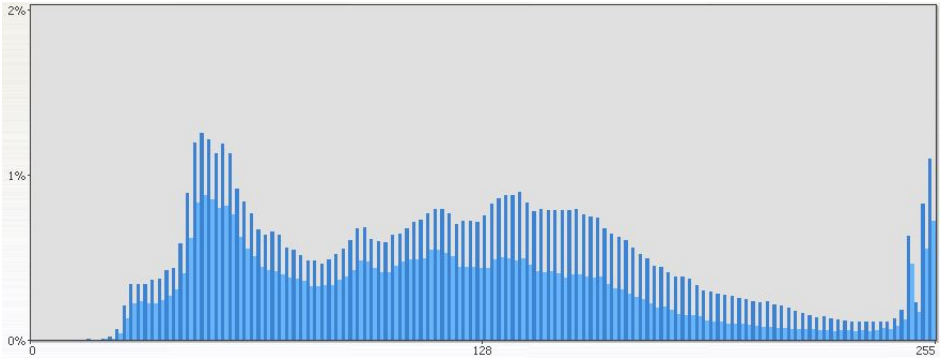

FIG. 11 – Histogramme fichier 100 KBytes.

Ces figures confirment que l'asymétrie est inversement proportionnelle à la taille du fichier dissimulé. Il sera donc plus aisé de détecter des contenus cachés de petites tailles, alors que

marge de manoeuvre sera plus étroite pour ce qui est des fichiers utilisant au maximum la capacité du conteneur.

L'analyse des écarts à l'aide du tableau 3 fait remarquer que la différence de distribution est notable même en ce qui concerne le cas limite d'une forte utilisation de la capacité du conteneur. Dans ce premier tableau, les données dissimulées n'ont pas été préalablement chiffrées. Les fichiers dissimulés sont tous des fichiers textes au format ASCII. Les caractères usuellement utilisés dans un fichier texte ont la particularité de comporter un 0 comme bit de point le plus fort. Cela implique un biais au niveau de la distribution des valeurs binaires. Cela aura l'effet d'accentuer l'écart entre les valeurs paires et impaires de byte.

Pour pouvoir déterminer de manière optimale l'écart type minimum entre ces catégories, il faut idéalement que le flux de données dissimulées au sein du fichier soit uniforme. Un tel flux de données peut être approché en chiffrant les données.

Données cachées [Bytes]	bytes pairs	bytes impairs	% pairs	% impairs	% écart
0	472'729	448'871	48.7	51.3	2.6
1	594	921'006	0	100	100
50'000	749'184	172'416	81.3	18.7	62.6
100'000	577'001	344'599	62.6	37.4	25.2

TAB. 3 – Distribution du plan LSB en fonction de la quantité de données dissimulées.

L'expérience est donc réitérée avec des données chiffrées avant dissimulation. Le tableau 4 permet de relever que les écarts se sont amoindris. La différence entre une image naturelle et une image stéganographiée à l'aide d'Invisible Secrets reste cependant notable. Au vue du tableau précité, un seuil de 10 % d'écart semble être adapté pour la détection du contenu stéganographique.

Données cachées [Bytes]	bytes pairs	bytes impairs	% pairs	% impairs	% écart
0	472'729	448'871	48.7	51.3	2.6
1	921'048	552	100	0	100
50'000	716'312	205'288	77.7	22.3	55.4
100'000	511'443	410'157	55.5	45.5	10

TAB. 4 – Distribution du plan LSB en fonction de la quantité de données dissimulées (cryptées).

Ce tableau met en évidence un comportement bien particulier du produit. Les informations concernant le fichier contenant uniquement 1 byte de donnée soulignent une inversion des proportions. Dans le cas du chiffrement préalable du fichier, contrairement à ce qui été opéré dans l'autre situation, les LSB non impactés par la dissimulation des données prennent la valeur de 1 au lieu de 0 précédemment. Il semble que le produit accentue de manière délibérée l'écart entre les groupes. Ce type de comportement semble être intentionnel. Des

considérations marketing peuvent être à l'origine de ce comportement à l'inverse des buts recherchés par la stéganographie. Ce produit gratuit a peut-être été bridé afin de ne pas faire de l'ombre à leur produit commercial du même nom. Cette spécificité du produit n'est pas clairement indiquée, pour un utilisateur non averti, elle sera donc invisible.

Un pseudo-code de détection de ce genre de modifications peut être élaboré à partir des observations précédentes. Une partie du code devra permettre de détecter que l'image n'est pas de type synthétique. Cette analyse statistique n'étant pas applicable dans ce cadre.

```
for(i=0; i < bytesImage.length; i++){
  cpt[bytesImage[i]]++;
}

for(i=0; i < cpt.length;i++){
  if (cpt[i]/bytesImage.length > 0.1) skipImage();
  else if (i % 2 == 0) cptPair += cpt[i];
  else cptImpair += cpt[i];
}

ecart = abs((cptPair - cptImpair)/bytesImage.length);

if (ecart >= 0.1) return true;
else return false;
```

4.6.2 Signature du produit

Dans la section précédente, une détection par analyse statistique de l'image a pu être développée afin de détecter les dissimulations opérées par Invisible Secrets 2.1. Malgré que cette analyse soit très efficace sur les images naturelles, elle a le gros désavantage d'être inutilisable sur les images synthétiques. Or ce genre d'image est largement représenté sur Internet. Il semble important de pouvoir détecter les manipulations indifféremment de la nature de l'image.

La mise en évidence d'une signature pourrait être une des méthodes permettant d'arriver à ce but. Une signature permet une détection très efficace avec une très faible probabilité de faux positifs (dépendant de la longueur de cette dernière). Contrairement aux analyses statistiques, dont certaines des propriétés peuvent se retrouver dans des fichiers normaux (non stéganographiés), une signature a très peu de chance d'être présente de manière native (sans avoir modifier le fichier).

Cette propriété dépend cependant d'un paramètre, la longueur de la signature. Il est aisément concevable qu'une signature de 32 bits soit un meilleur indicateur qu'une signature de 8 bits. Mathématiquement parlant, une signature sur 8 bits revient à $\frac{1}{2^8} = 0.39\,\%$ de chance de tombé sur cette signature. Cela peut paraître minime, cependant relativement à la taille d'un fichier BMP (approchant régulièrement le million de Bytes), il n'en est pas ainsi. Comparativement, une signature de 32 bits représente une probabilité de $\frac{1}{2^{32}} = 2.91 * 10^{-11}$. Il en revient donc que plus cette signature sera longue, plus le résultat pourra être considéré comme fiable.

Cependant, dans la pratique tout produit de stéganographie qui se respecte essayera d'éviter de laisser ce genre de preuve derrière lui. Il en résulte donc que la mise en évidence d'une signature ne sera pas possible pour chaque produit, ou du moins représentera une tâche ardue.

Dans le cadre d'Invisible Secrets 2.1, l'analyse minutieuse des fichiers a pu révéler un trait commun au niveau de chaque fichier. Pour en arriver à ce point, il a fallu commencer par extraire les données représentées par les bits de poids faible de l'image. Le fichier dissimulant uniquement 1 bytes a été choisi pour commencer les recherches. Si l'on se souvient des observations effectuées dans la section précédente, les bits LSB des bytes n'ayant pas servi à la dissimulation ont tous la même valeur. Cela permet de facilement réduire la zone de recherche de signature.

La première observation est que pour un fichier de 1 Byte, la quantité de données modifiée équivaut à 153 Bytes. Des données supplémentaires au seul fichier dissimulé ont donc été ajoutées. Cela est de bonne augure concernant la présence d'une éventuelle signature. De plus on peut remarquer que les données sont concentrées en une zone compact. Les données n'ont pas été disséminées à travers tout le fichier. Cela ne permet cependant pas de prétendre que la modification a été faite de manière séquentielle.

En analysant cette zone de plus près, on remarque la présence d'une suite de caractères bien particulière (voir figure 12). Cette chaîne de caractères correspond à la suite hexadécimale `AA AA AA AA 0F 74 6D 70.` L'analyse de 4 autres fichiers montre que cette suite est présente dans chacun des cas (fichier chiffré ou clair, compressé ou pas).

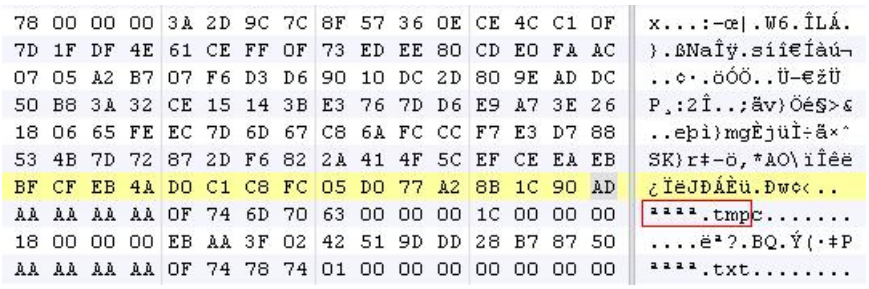

FIG. 12 – Suite de caractères intéressante dans les données dissimulées par Invisible Secrets 2.1.

Cette suite peut donc être utilisée comme signature du logiciel.

4.6.3 Probabilité de faux positifs

En se basant sur les critères mentionnés plus haut, la taille de cette signature est de 64 bits. La probabilité de retrouver cette même suite par hasard dans un fichier est très faible, de l'ordre de $5 * 10^{-18}$ %. L'analyse par signature de Invisible Secrets est théoriquement très fiable.

Pour ce qui est de l'analyse statistique, il est difficile de donner une approximation théorique de la probabilité de faux positifs. Afin de pouvoir donner une bonne estimation de cette probabilité, il faudrait verifier l'uniformité de distribution sur plusieurs milliers d'images.

Le pourcentage d'images déviant de cette hypothèse renseignera sur la probabilité de faux positifs de la méthode.

Par l'union des deux analyses présentées ci-dessus, une méthode de détection fiable des dissimulations effectuées sur des fichiers BMP à l'aide de Invisible Secrets 2.1 est envisageable. La découverte de la signature peut être suffisante de par sa grande fiabilité comme mentionné précédemment. L'analyse statistique apporte cependant un élément de preuve supplémentaire d'un usage suspect d'une image numérique. Cela conforte les résultats fournis par la détection de signature. Pour finir, un tableau (5) récapitulatif des performances de chacune des analyses.

Analyse	Nbre de détections	Nbre de faux positifs	Efficacité [%]	Faux positifs [%]
Statistique	14	3	64.7	21.4
Signature	17	0	100	0

TAB. 5 – Performance des analyses pour Invisible Secrets 2.1.

4.7 WbStego4Open

WbStego4Open est un logiciel Open source. L'étude de ce logiciel est intéressante car il est disponible sur Windows, mais aussi sur plateforme Linux. Le produit a été développé en Delphi. De par sa nature Open source, les sources du produit sont disponibles au téléchargement. Il permet la dissimulation dans 4 types de fichiers différents :

- Windows Bitmap (BMP)
- Fichier texte (TXT)
- Adode Portable Document Format (PDF)
- Fichier HTML

L'analyse se focalise sur le format BMP. Pour ce dernier, les fichiers de 4, 8 et 24 bits sont supportés. Les fichiers Bitmap 4 bits n'étant pas vraiment courants, le fonctionnement de la dissimulation sur ce type de fichier n'est pas couvert. Les fichiers 8 bits ne sont également pas couverts. Malgré que la description du produit mentionne la compatibilité de ce type de fichier, en pratique, il est impossible d'y dissimuler des données. L'opération s'achève de façon prématurée par l'affichage d'un message d'erreur (figure 13), peut importe la taille du fichier à camoufler.

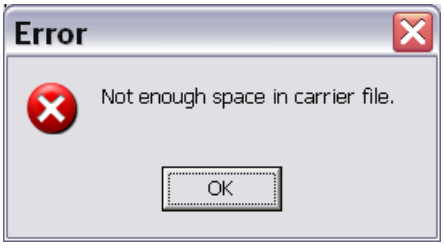

FIG. 13 – Fichier 8 bits impossible à utiliser comme conteneur.

Le produit laisse le choix de chiffrer ou non les données, il laisse aussi le choix au niveau de la dissimulation du fichier au travers du conteneur. Cette opération peut être effectuée de

manière séquentielle en partant depuis le début du fichier, ou encore disséminée à travers le conteneur. Cela donne 4 cas généraux à examiner :

- Séquentiel
  - Chiffré
  - Non Chiffré
- Distribué
  - Chiffré
  - Non Chiffré

L'écriture séquentielle des données va d'abord être analysée. Avant de commencer l'analyse du produit en lui même, il est important de souligner quelques détails concernant l'utilisation d'un algorithme de dissimulation séquentiel. Pour permettre le décodage des données cachées, le logiciel doit connaître plusieurs éléments ; il s'agit de l'offset par rapport au début de l'image et la taille des données. Ces deux informations constituent le minimum nécessaire pour bien décoder l'information. A noter que en ce qui concerne la taille des données, deux implémentations sont possibles : Inscrire la taille des données à un endroit précis dans le fichier, définir une chaîne de caractères spécifiant la fin des données.

La comparaison du fichier résultant de la manipulation avec le fichier conteneur, montre que les modifications ont lieu en commençant au début du fichier. Aucun offset n'est ajouté, les données commencent au début des données d'images (après l'en-tête et la palette Bitmap). L'extraction du dernier plan de bit met en évidence un détail intéressant. Suivant les trois premiers bytes, l'extension du fichier caché est ajouté par le programme. Les données commencent directement après cette information. L'extension étant contenue dans les bytes 4-6 au niveau des LSB, les bytes 1-3 sont peut-être aussi porteurs d'information. La comparaison de ces 3 premiers bytes en modifiant la taille du fichier cachées montre un liens direct. Cependant, la taille du fichier n'est pas écrite de manière normale. Elle est ajoutée en début de fichier, mais en commençant par les bytes de poids faible (figure 14).

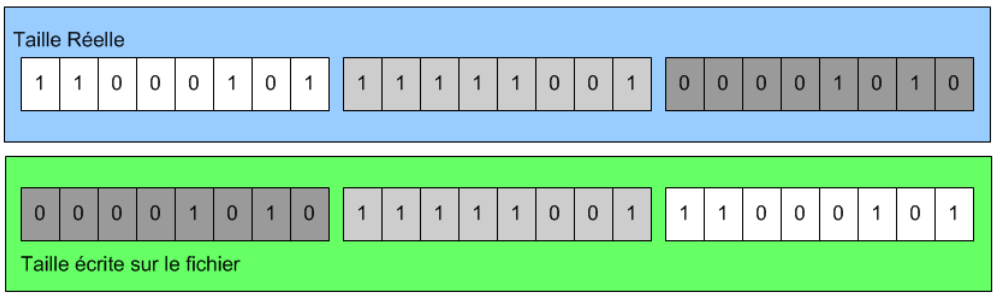

FIG. 14 – Représentation de la taille dans le fichier dissimulé par WbStego.

Dans le cas d'un préalable chiffrement des données avant dissimulation, le fonctionnement diffère légèrement. La taille des données est toujours écrite de la même manière, mais en lieu et place de l'extension du fichier, le programme place la valeur hexadécimale `01 FF 01` lui indiquant que le fichier qui suit est chiffré. La valeur est indépendante de l'algorithme de chiffrement utilisé. Pour résumé, la figure 15 montre le format appliqué par le produit dans le cadre d'une dissimulation séquentielle des données.

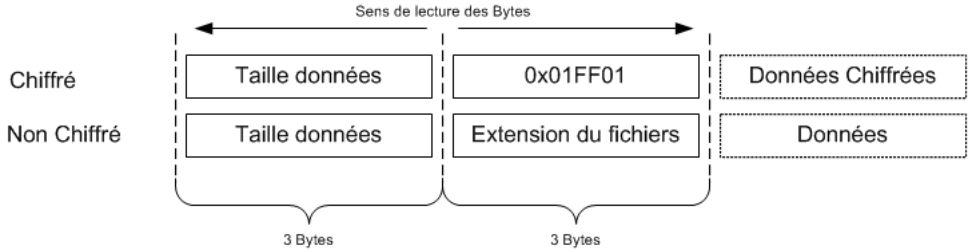

FIG. 15 – Format des données dissimulées par wbStego en mode séquentiel.

Pour une dissimulation « distribuée » à travers le conteneur, les paramètres nécessaires lors de la dissimulation séquentielle ne sont pas forcement applicables. La taille des données va certainement se retrouver insérée au niveau des données. Cependant, selon la méthodologie choisie, elle sera peut-être placée de manière aléatoire au sein du fichier. Pour pouvoir effectuer la distribution, un élément doit être choisi afin de paramétrer cette dernière. Cela peut se faire de plusieurs manières. Il est par exemple possible de définir un saut fixe au niveau des données. Par exemple définir que les données seront insérées tous les 5 bytes de données d'image. Une fois arrivé à la fin du fichier, la dissimulation continue au début du fichier en ayant préalablement décalé la position de 1. Cette solution est une manière sommaire de distribués les données à travers le fichier. La méthode la plus utilisée pour ce genre d'opération est la génération d'une suite de nombres aléatoires indiquant la position ou dissimulé l'information dans l'image. La propriété essentielle d'une telle suite est de pouvoir être recréée à l'identique lors du décodage de l'information cachée.

Dans le cadre de wbStego, aucune saisie d'information n'est demandée afin de définir cette suite (pas de saisie de passphrase demandée). Le programme peut donc agir de trois manières différentes :

- Distribution statique des données
- Création d'une suite de nombres pseudo-aléatoires en se basant sur des informations figées (taille du conteneur, etc...)
- Création d'une suite de nombres pseudo-aléatoires en utilisant une information non figée, mais écriture de cette information dans le fichier

L'analyse du fichier montre que la taille des données embarquées a pris la place de l'extension. Elle a été insérée dans le fichier de la même manière que précédemment (figure 14). Il a été remarqué que les trois premiers Bytes, dans le cas d'une distribution des données, représentent à peu près la contenance maximal du conteneur. Ce fait a pu être vérifié de manière empirique. L'extension, quand à elle, se retrouve distribué dans le fichier. Il n'est plus possible de se baser sur sa présence car, au travers de plusieurs tests, il a été remarqué que la distribution n'était pas statique le long du fichier. Malgré cela les éléments précédents suffisent à la définition d'une analyse.

Pour résumer, l'analyse se base sur plusieurs éléments afin de pouvoir déterminer si le fichier a été utilisé comme conteneur dans WbStego. Premièrement, il est vérifié que la taille (wbStego) formée par les trois premiers Bytes n'est pas plus grande que la capacité maximale du conteneur. Ensuite plusieurs situations sont envisagées. Si la dissimulation est séquentielle, il faut vérifier que l'extension corresponde à un type de fichier connu, ou vérifier

la présence de la signature de fichier crypter `0x01FF01`. Dans le cas d'une dissimulation distribuée dans le fichier, la seconde taille est vérifiée. Elle doit être plus petite que la capacité maximal du conteneur, mais aussi plus petite que la précédente taille. La figure 16 résume la condition sur chaque champ.

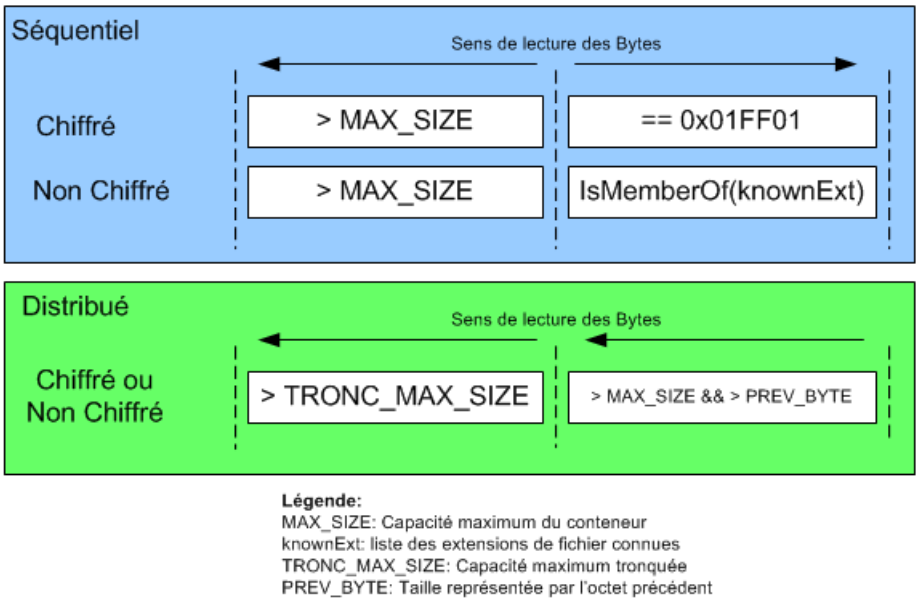

FIG. 16 – Tests effectués pour l'analyse de WbStego.

4.7.1 Probabilité de faux positifs

Un chiffre théorique est difficile à donner. Contrairement aux autres analyses, ici la détection n'a pas lieu sur une signature statique. Chaque champ contenant des données variables, une estimation de la probabilité de faux positifs est difficile à calculer. Il faut se fier aux chiffres émanant de la pratique, pour pouvoir quantifier cette valeur.

L'expérimentation à l'aide du script développé donne de très bons résultats. Parmi les 300 fichiers Bitmap composant la base de référence, aucun n'a été faussement détecté. Sur cette base, l'analyse n'engendre pas de faux positifs. Pour ce qui est de la capacité de détection, un seul fichier n'a pas été détecté par l'analyse. Il s'agit d'un fichier de 1 Byte dissimulé de manière distribuée dans le fichier. De plus, le produit lui même n'a pas été capable de récupérer les données se trouvant dissimulés dans ce fichier. Il s'agit donc certainement d'une erreur d'implémentation.

4.8 Hiderman

Hiderman fait partie de ces produits permettant d'utiliser n'importe quel format comme conteneur de contenu stéganographique. Ce genre de possibilité donne beaucoup d'information concernant le type de méthode stéganographique utilisée par ce logiciel.

Le logiciel se contente d'ajouter les données à la fin du fichier conteneur, sans même se soucier si cela aurait un impact sur le conteneur. Pour la plupart des fichiers, l'adjonction de données à la fin du fichier n'a aucune influence. Le BMP, JPEG, PNG, WAV sont autant de formats qui ne posent aucun problème. Cela n'est pas le cas du format MP3 par exemple. La version 1 des tags ID3 a été définie pour être lue à partir de la fin du fichier. L'adjonction de données rend donc illisible ces tags (figure 17).

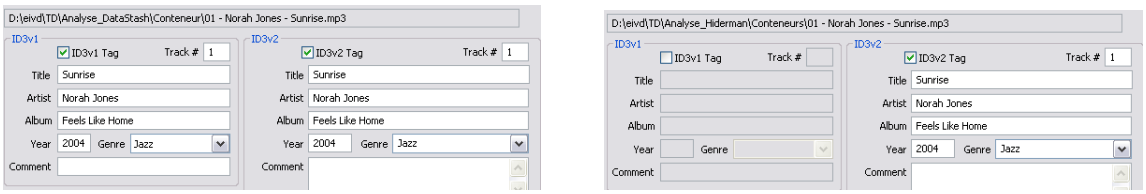

FIG. 17 – A Gauche, tags du fichier original. A droite, tags sur le fichier stéganographié.

Dans le cadre du MP3, cela n'a pas de lourdes conséquences. Tout d'abord, le fichier reste utilisable. Ensuite, les tags ID3 en version 2 sont placés en début de fichier (figure 17). Sous condition d'avoir un lecteur permettant la lecture des ID3v2, aucun changement ne sera perçu par l'utilisateur.

Cependant, cette exemple montre clairement qu'aucun test n'est effectué préalablement. Le programme ne se soucit pas de savoir si son utilisation va rendre le conteneur inutilisable. L'utilisation sur tout type de format n'est pas réellement applicable.

En se basant sur les informations trouvées dans [23], Hiderman marque d'une signature les données traitées. Cette signature se trouve après les données dissimulées. Elle correspond au caractère CDN en codage ASCII (figure 18).

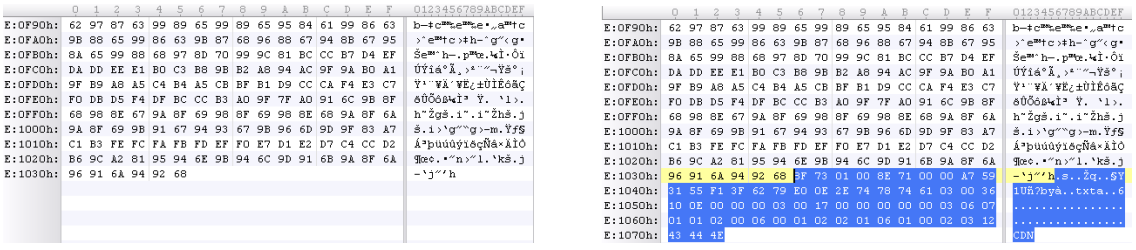

FIG. 18 – A gauche, la fin du fichier original. A droite, le fichier contenant des données.

L'utilisation de plusieurs conteneurs différents permet de confirmer que la signature n'est pas dépendante du format utilisé. Il n'est pas important de différentier le format du fichier lors de la recherche de signature. Une analyse générique cherchant la signature du produit en fin de fichier est suffisante.

4.8.1 Probabilité de faux positifs

La taille de la signature étant de 24 bits, la probabilité de faux positif est de $\frac{1}{2^{24}}$. La signature étant apposée à un endroit précis dans le fichier, cette probabilité est encore plus faible.

L'analyse développée en EnScript détecte la présence de la signature Hiderman sur tout type de fichier. Appliquée à la base de référence, l'analyse se montre très efficace. Un taux de détection de 100 % sans aucun faux-positif est observé.

4.9 Data Stash 1.5

Data Stash est édité par une société de Singapour du nom de SkyJuice Software. Une licence coûte 39$. Ce produit permet l'utilisation de n'importe quel fichier comme conteneur. Guillermito [16] a déjà fait une analyse de ce produit dans sa version 1.1b. A l'instar de Hiderman, Data Stash se contente d'ajouter les données à dissimuler en fin de fichier. L'analyse de Guillermito a même démontrer que, contrairement à ce que mentionne le descriptif du produit, les données ne sont nullement chiffrées. Un simple éditeur hexadécimal peut être utilisé pour retrouver les données cachées.

Pour la version 1.5 du produit, les choses ont un peu été revues. Le principe de fonctionnement du programme est resté le même. Cependant, il n'est plus possible d'extraire les données aussi aisément. Sans pouvoir prouver que les données sont cette fois-ci chiffrées, il est clair qu'elles n'ont pas simplement été écrites de manière séquentielle en fin de fichier.

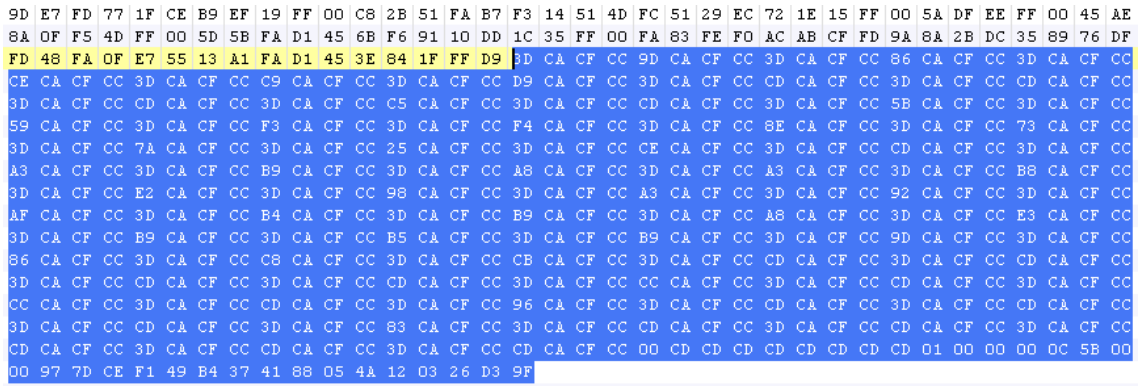

FIG. 19 – Données dissimulées avec Data Stash 1.5.

Des bytes en fin de fichier restent inchangés, indépendamment des formats du couple conteneur-contenu, ainsi que le mot de passe saisi lors de la dissimulation. Deux parties peuvent être distinguées. Elles se trouvent séparées par trois bytes dépendant du format du conteneur. Cela donne en hexadécimal :

1. `01 00 00 00 XX XX XX 00 97 7D CE F1 49 B4 37 41 88 05 4A 12 03 26 D3 9F`

Les XX correspondent aux bytes n'étant pas statiques. La longueur de cette signature est conséquente, cela permet d'assurer une faible probabilité de faux positifs.

Si aucun mot de passe n'est saisi lors de la dissimulation, cette séquence de bytes se trouve précédée par la séquence suivante :

1. `00 CD CD CD CD CD CD CD CD`

Cette deuxième signature permet de déterminer si une attaque par dictionnaire ou force brute est nécessaire. Dans le cas contraire, l'utilisation du produit suffit à pouvoir récupérer les données dissimulées.

4.9.1 Probabilité de faux positifs

Étant donné la taille de la signature de l'application, la probabilité de détecter un faux positif est très faible. La signature servant fait 21 bytes, soit 168 bits (en ne comptant pas les 3 bytes dépendant du type de fichier). Il y a donc $\frac{1}{2^{168}}$ de chance que le fichier détecté soit un faux positif. Cette probabilité diminue encore lorsque aucun mot de passe n'est saisi lors du processus de dissimulation. Dans ce cas de figure, la probabilité de faux positif est de $\frac{1}{2^{240}}$. De plus cette signature étant positionnée à un endroit bien précis au sein du fichier, cette probabilité est d'autant plus faible.

Dans la pratique, aucun fichier n'est détecté comme faux positif. Une détection à 100 % des fichiers stéganographiés est observée.

La détection par signature de Data Stash 1.5 est donc très efficace. Un fichier détecté par cette analyse à une probabilité très forte d'être utilisé comme conteneur stéganographique.

5 Organisation du travail

Ce travail s'est déroulé durant la période allant du 15 Septembre au 15 Décembre 2008. Il consistait en 12 semaines effectives de travail. Il a été précédé d'un pré-projet s'étant déroulé durant le dernier semestre de cours. Ce dernier a eu pour objectif de dégrossir le sujet afin de mieux comprendre ce qu'était la stéganographie.

La partie portage de l'existant en EnScript a duré 6 semaines. Elle était initialement prévue sur 1 mois, mais des problèmes lors de l'adaptation de certaines parties du code, tel que le décodeur JPEG, ont finalement allongé de deux semaines cette échéance. Quelques jours supplémentaires ont été nécessaires à la correction de différents bugs. Ces jours ont été répartis le long du travail de diplôme.

La partie écriture de nouvelles analyses n'a pas bénéficié de planification précise. La nature de cette tâche rendait difficile l'estimation d'une durée finie pour l'accomplir. Selon le produit, l'analyse pouvait prendre plusieurs jours, sans compter l'implémentation. De plus, certaines analyses n'ont pas été concluantes. L'analyse du produit Steganos a duré une semaine, sans qu'aucune moyen de détection ne puisse être trouvé. 5 semaines ont donc été attribuées à la définition et l'implémentation de nouvelles analyses.

La dernière semaine a été attribuée à l'écriture et la correction du présent document.

Les tests et la validation de l'application ont été menés durant les deux dernières semaines de travail de diplôme.

6 Conclusion

La stéganographie est un domaine passionnant, et son étude s'est révélée extrêmement intéressante. Chaque méthode de dissimulation requiert une étude particulière, la monotonie n'a pas le temps de s'installer. De plus, cette discipline nécessite d'avoir des connaissances dans des domaines très variés, comme la statistique, l'imagerie et l'audio numériques et, pour l'implémentation, des connaissances en programmation.

Mais malgré sa grande diversité, il est regrettable que les acteurs actuels du marché se limite à utiliser toujours les mêmes méthodes. Du point de vue de la détection, cela est très bénéfique. L'analyse ayant été développée pour un certain produit peut être réutilisée pour l'élaboration d'autres analyses. Cela facilite grandement la tâche du stéganalyste. Mais d'un point de vue purement scientifique, cette attitude nuit beaucoup au développement de ce domaine, que ce soit pour la stéganographie ou la stéganalyse.

En ce qui concerne l'implémentation de l'application, elle a pu être portée et étendue avec succès en EnScript. Cette phase n'a cependant pas été anodine, de part les nombreuses différences syntaxique avec le langage de base qu'était le JAVA. Malgré que l'étape se soit déroulée avec succès, il semble que, dans l'état actuel, le EnScript ne soit pas à recommander pour ce type de développement. Cependant, si l'on se contente d'analyse de signature ne nécessitant pas la lecture complète du fichier, le EnScript est relativement performant dans ce domaine. Si l'application devant être développée va au delà de cela, avec notamment l'implémentation d'analyses statistiques, le choix d'un autre langage semble plus judicieux. On peut même éventuellement utiliser des langages de haut niveau spécialisé dans le calcul, tel que MATLAB, permettant l'implémentation facilitée des nombreuses analyses statistiques existantes.

Toutefois, le langage EnScript est en constante évolution. Chaque nouvelle version de EnCase apporte de nombreuses améliorations. Pour l'instant il est encore difficile de définir l'utilisation fait de la stéganographie, et dans ces conditions l'amélioration du langage pour ce domaine d'application ne se fait pas ressentir. La stéganographie étant une science du secret, quantifier son utilisation est très difficile. Plus l'offre concernant des programmes de détection sera étoffée, plus une réponse précise pourra être donnée à cette question.

D'un point de vue personnel, ce travail m'a permis d'agrandir mes connaissances sur le sujet. L'élaboration de nouvelles analyses est une activité vraiment intéressante. La mise au point d'un script permettant la détection de données à priori invisible m'a procuré beaucoup de satisfaction.

Ljupce Nikolov

Deuxième partie

Documentation utilisateur

7 Description de l'application

L'application permet de parcourir une arborescence de fichier, à la recherche de contenu stéganographié. Elle a été développée pour l'application EnCase, en utilisant le langage de script intégré, le EnScript. Ci-dessous, une figure rappelant le principe de fonctionnement de l'application.

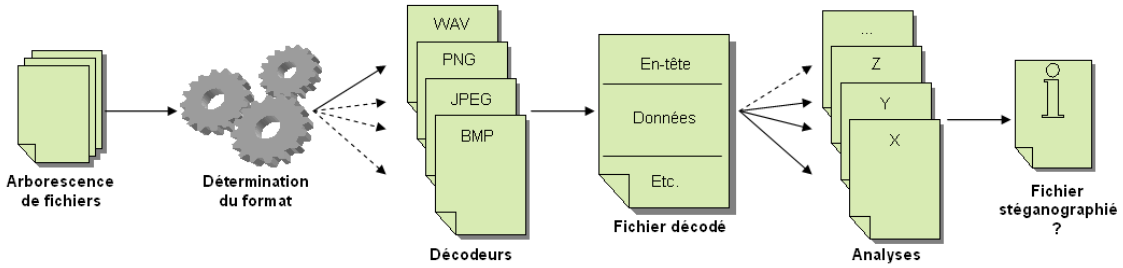

FIG. 20 – Principe de fonctionnement.

8 Pré-requis

Le script nécessite uniquement la présence de EnCase dans sa version 6 pour fonctionner. Le script a été développé sur la version 6.12 du produit. Son fonctionnement sur des versions ultérieures ne peut pas être garanti.

9 Installation du Script

Le script ne nécessite pas d'installation à proprement parler. Il suffit de le copier dans le répertoire contenant la collection de EnScript. Le chemin de ce répertoire correspond à `%ENCASE_DIRECTORY%\EnScript\`. Pour une installation standard, `%ENCASE_DIRECTORY%` correspond à `C :\Program Files\EnCase6\` pour une machine sous Windows XP.

Un sous-dossier contenant le script pourra être crée à cet endroit. Au lancement de EnCase, le script sera présent dans la vue « EnScript ».

10 Interface utilisateur

L'interface utilisateur de l'application est principalement composée de deux éléments : La fenêtre de configuration du script et la vue de présentation des résultats.

10.1 Fenêtre de configuration du script

Cette fenêtre est l'élément central de l'application. Elle permet de paramétrer son fonctionnement. Elle est composée de plusieurs parties dont l'élément principal est un arbre présentant les analyses pouvant être menées.

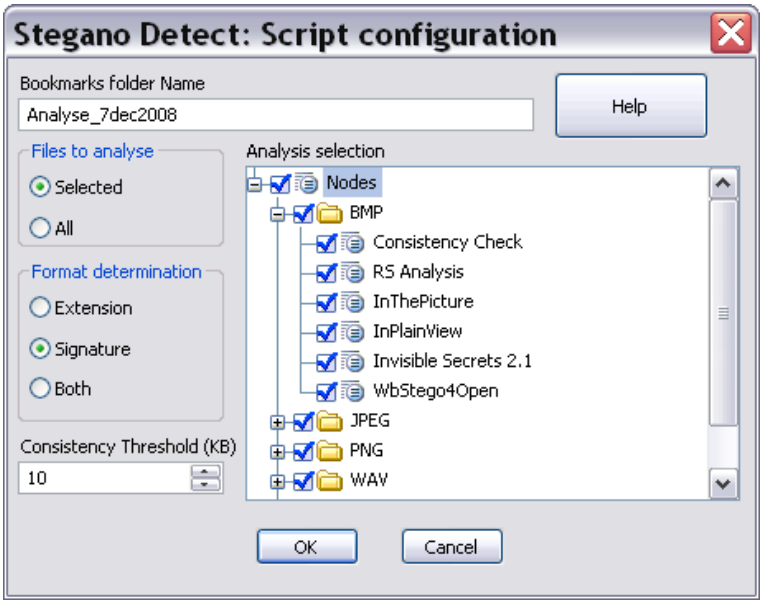

FIG. 21 – Fenêtre de configuration des analyses.

Bookmarks folder Name Permet de spécifier le nom du dossier racine de l'arborescence de résultat.

Files to analyse Permet de limiter l'analyse au seuls fichiers sélectionnés, ou au contraire, analyser tous les dossiers présents dans le cas EnCase.

Format determination Donne le choix de la méthode utilisée pour déterminer le format du fichier. **Extension** signifie que seul l'extension du fichier est prise en compte pour identifier le format. Cela peut engendrer que de nombreux fichiers soient définis comme n'ayant pas été traités. **Signature** se base sur la base de signature présente dans EnCase pour déterminer le format du fichier. Cette méthode est plus complète, mais aussi plus lente. Enfin, avec l'option **Both**, seul les fichiers dont l'extension et la signature coïncident seront analysés.

Consistency Threshold Utilisé par les analyses de consistance, ce paramètre permet de spécifier la taille (en kilobytes) à partir de laquelle les données contenues dans des endroits inappropriés sont vues comme positif. Ce paramètre peut être saisi entre 1 et 100 kilobytes.

Analysis selection Liste les analyses développées. Elles sont regroupées par format pris en charge par l'analyse. Celles présentent dans **General analysis** peuvent être appliquées à n'importe quel format de fichier.

Help Ce bouton affiche une fenêtre résumant la fonctionnalité de chaque analyse.

10.2 Présentation des résultats

Les résultats sont affichés dans la vue Bookmarks dans EnCase. Le répertoire racine, dont le nom a été spécifié lors de la configuration de l'application, est formé de trois sous-dossiers : Suspicious Content, Apparently not suspicious content et unprocessed files. Chacun des fichiers analysés se retrouve dans un de ces trois sous-dossiers.

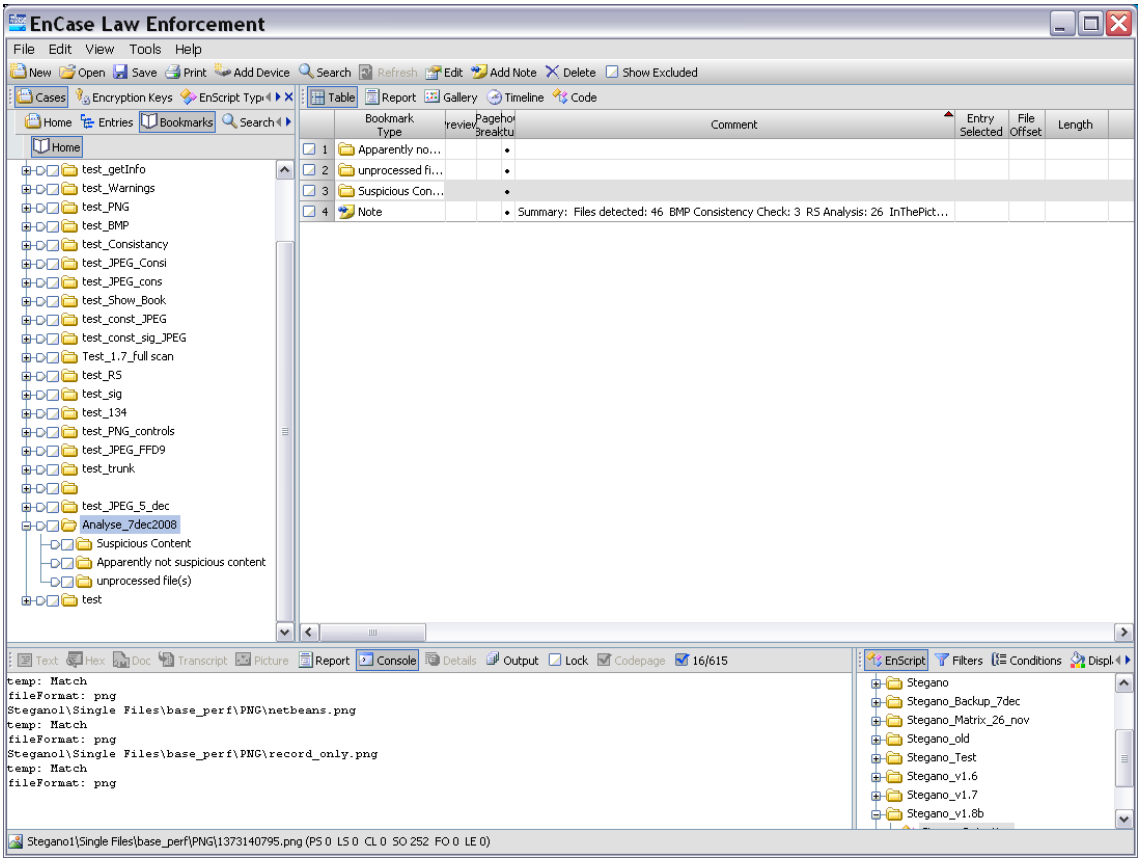

FIG. 22 – Vue Bookmarks dans EnCase.

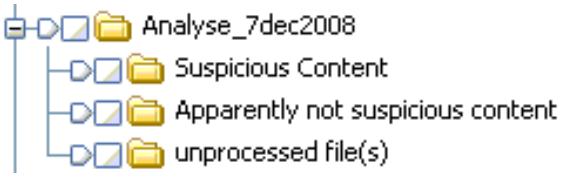

FIG. 23 – Arborescence des résultats.

Suspicious Content Contient les fichiers présentant certaines caractéristiques de conteneur stéganographique.

Apparently not suspicious content Regroupe les fichiers n'étant, d'après les analyses demandées, pas vus comme conteneur stéganographique.

Unprocessed file(s) Contient les fichiers n'ayant pas pu être traités par les décodeurs. Se retrouvent ici, les fichiers n'ayant pas le bon format, les fichiers tronqués ainsi que les fichiers n'étant pas gérés par l'application (voir Limitations).

Afin de pouvoir retrouver rapidement les résultats d'une analyse, une copie du contenu de la fenêtre de résumé est sauvegardé à la racine du dossier. Les détails concernant ces résultats sont insérés dans la zone commentaire des Bookmarks.

	Bookmark Type	Comment	Name	In Report	File Ext	File Type	File Category	Signature
12	Notable File	JPEG: Unused data at the end of the file (77 kb)	foret_oursecret.jpg	•	jpg	JPEG	Picture	Match
13	Notable File	JPEG: Unused data at the end of the file (77 kb)	foret_oursecret.jpg	•	jpg	JPEG	Picture	Match
14	Notable File	JPEG: Unused data at the end of the file (156 kb)	foret_stealth.jpg	•	jpg	JPEG	Picture	Match
15	Notable File	JPEG: Unused data at the end of the file (156 kb)	foret_stealth.jpg	•	jpg	JPEG	Picture	Match
16	Notable File	WbStego signature detected: plaintext sequential mode (extension = txt)	House_wbStego_input_audio.bmp	•	bmp	Bitmap Image	Picture	Match
17	Notable File	WbStego signature detected: distributed mode	House_wbStego_input_audio_evenly_no_crypt...	•	bmp	Bitmap Image	Picture	Match
18	Notable File	File is positive to RS Analysis: message Length: 0.95115954324179708	House_wbStego_input_audio_evenly_no_crypt...	•	bmp	Bitmap Image	Picture	Match
19	Notable File	WbStego signature detected: plaintext sequential mode (extension = jpg)	House_wbStego_jpeg_BookmarkView.bmp	•	bmp	Bitmap Image	Picture	Match
20	Notable File	File is positive to RS Analysis: message Length: 0.30139959148891515	House_wbStego_jpeg_BookmarkView.bmp	•	bmp	Bitmap Image	Picture	Match
21	Notable File	WbStego signature detected: ciphertext sequential mode	house_wbstego_jpeg_bookmarkview_crypt.bmp	•	bmp	Bitmap Image	Picture	Match
22	Notable File	File is positive to RS Analysis: message Length: 0.32535391782324541	house_wbstego_jpeg_bookmarkview_crypt.bmp	•	bmp	Bitmap Image	Picture	Match
23	Notable File	File is positive to RS Analysis: message Length: 0.32589163743294858	house_wbstego_jpeg_bookmarkview_crypt_de...	•	bmp	Bitmap Image	Picture	Match
24	Notable File	WbStego signature detected: ciphertext sequential mode	house_wbstego_jpeg_bookmarkview_crypt_de...	•	bmp	Bitmap Image	Picture	Match
25	Notable File	WbStego signature detected: distributed mode	house_wbstego_jpeg_bookmarkview_nocrypt_...	•	bmp	Bitmap Image	Picture	Match
26	Notable File	File is positive to RS Analysis: message Length: 0.85719581129618883	house_wbstego_jpeg_bookmarkview_nocrypt_...	•	bmp	Bitmap Image	Picture	Match
27	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_100k_nocomp_cryp.bmp	•	bmp	Bitmap Image	Picture	Match
28	Notable File	File is positive to RS Analysis: message Length: 0.7683141840903176	neige_100k_nocomp_cryp.bmp	•	bmp	Bitmap Image	Picture	Match
29	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_100k_nocomp_nocryp.bmp	•	bmp	Bitmap Image	Picture	Match
30	Notable File	File is positive to RS Analysis: message Length: 0.70950382873341999	neige_100k_nocomp_nocryp.bmp	•	bmp	Bitmap Image	Picture	Match
31	Notable File	File is positive to RS Analysis: message Length: 1	neige_100k_nocomp_nocryp_LSB.bmp	•	bmp	Bitmap Image	Picture	Match
32	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_1_nocomp_cryp.bmp	•	bmp	Bitmap Image	Picture	Match
33	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_1_nocomp_nocryp.bmp	•	bmp	Bitmap Image	Picture	Match
34	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_50k_nocomp_cryp.bmp	•	bmp	Bitmap Image	Picture	Match
35	Notable File	File is positive to RS Analysis: message Length: 0.40660285353083603	neige_50k_nocomp_cryp.bmp	•	bmp	Bitmap Image	Picture	Match
36	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_50k_nocomp_nocryp.bmp	•	bmp	Bitmap Image	Picture	Match
37	Notable File	File is positive to RS Analysis: message Length: 0.39733889398369598	neige_50k_nocomp_nocryp.bmp	•	bmp	Bitmap Image	Picture	Match
38	Notable File	File is positive to RS Analysis: message Length: 0.64060639786214402	neige_bookmark_invisible_comp_cryp.bmp	•	bmp	Bitmap Image	Picture	Match
39	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_bookmark_invisible_comp_cryp.bmp	•	bmp	Bitmap Image	Picture	Match
40	Notable File	File is positive to RS Analysis: message Length: 0.64266954816822341	neige_bookmark_invisible_comp_nocryp.bmp	•	bmp	Bitmap Image	Picture	Match
41	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_bookmark_invisible_comp_nocryp.bmp	•	bmp	Bitmap Image	Picture	Match
42	Notable File	Invisible Secrets 2.1 Statistical and signature detected	neige_input_audio_invisible_comp_cryp_dede....	•	bmp	Bitmap Image	Picture	Match

FIG. 24 – TableView des Bookmarks dans « Suspicious Content ».

La section suivante donne des informations quant à l'interprétation de ces résultats.

10.3 Interprétation des résultats

Plusieurs types de commentaires ont été définis lors de l'affichage des résultats dans la vue « Bookmarks » de EnCase. En voici une liste exhaustive, avec l'interprétation qu'il faut en tirer.

Message	Interprétation
Apparently clean	Aucune des analyses n'a permis de détecter la présence de contenu stéganographié. Cependant, cela ne veut pas dire qu'aucun contenu n'est réellement dissimulé.
BMP : Compressed BMP	Le fichier est un BMP compressé. Ce type de fichier n'est pas pris en charge par le décodeur. Le fichier n'a pas été analysé.
BMP : Not 8 or 24 bits Bitmap	Le fichier à une profondeur de couleurs différente de celles prisent en charge par le décodeur. Le fichier n'a pas été analysé.

Message	Interprétation
BMP : Truncated Bitmap File	Le fichier BMP est incomplet. La lecture s'est terminée de manière abrupte. Le fichier n'a pas été analysé.
BMP : Unsupported Bitmap format	Le fichier est de type OS/2 non supporté par le décodeur. Le fichier en question n'a pas été analysé.
BMP : Unused data at the end of the file (N kb)	N kilobytes de données non utilisées pour l'affichage de l'image sont présentes en fin de fichier.
BMP : Unused data in file (header and end of file, total of N kb)	N kilobytes de données non utilisées, réparties entre l'en-tête et la fin du fichier, sont présentes dans le fichier.
BMP : Unused data in header (N kb)	N kilobytes de données non utilisées sont présentes entre l'en-tête et le début de l'image.
Data Stash 1.5 signature found : A password has been set	La signature de Data Stash 1.5 a été détectée en fin de fichier.
Data Stash 1.5 signature found : No password	La signature de Data Stash 1.5 a été détectée en fin de fichier. La signature correspondant à un fichier en clair est aussi présente.
Hiderman signature found	La signature caractéristique de Hiderman a été trouvée en fin de fichier.
InPlainView signature found	La signature de InPlainView a été détectée en début du fichier.
InThePicture signature found	La signature de InThePicture a été détectée en début du fichier.
Invisible Secrets 2.1 signature detected	La signature d'Invisible Secrets 2.1 a été détectée au sein du fichier.
Invisible Secrets 2.1 Statistical and signature detected	Les caractéristiques statistiques ainsi que la signature d'Invisible Secrets 2.1 ont été détectées dans le fichier.
Invisible Secrets 2.1 Statistical detection	Les caractéristiques propres à une dissimulation par Invisible Secrets 2.1 sont présentes dans le fichier.
JPEG : Error while reading data	Une erreur est survenue lors du décodage du fichier JPEG. Cela est généralement dû à un fichier incomplet. Le fichier n'est pas analysé.
JPEG : File is not in JPEG format	La signature `0xFFD8` du fichier JPEG n'est pas présente en début de fichier. Le fichier n'est pas analysé.
JPEG : File is not sequential	Le décodeur gère uniquement le fichier JPEG de type séquentiel. Les autres types ne sont pas décodés. Le fichier n'est pas analysé.
JPEG : Unknown format	Le fichier n'est pas de format JFIF. Il n'est pas supporté par le décodeur. Le fichier n'est pas analysé.

Message	Interprétation
JPEG : Unknown marker	Un marqueur JPEG inconnu est présent dans le fichier. La lecture ne peut pas continuer. Le fichier n'est pas analysé.
JPEG : Unused data at the end of the file (N kb)	N kilobytes de données inutilisées sont présentes après le marqueur `0xFFD9` de fin de fichier JPEG.
JPEG : Unused data in Comments (N kb)	N kilobytes de données sont présentes dans les commentaires.
JPEG : Unused data in file (comments and end of file, total of N kb)	N kilobytes de données, reparties entre la fin du fichier et les commentaires, sont présentes au sein du fichier.
Jsteg signature found	La signature JStegShell est présente dans le fichier.
PNG : Could not find the PNG header	Le bloc d'en-tête ne suit pas la signature PNG. Le fichier n'est pas analysé.
PNG : Data present in textfields (N kb)	N kilobytes de données sont présentes dans les différents blocs textuels.
PNG : Length of data chunk is not valid	La longueur du bloc de données ne correspond pas a la valeur définie. Le fichier PNG est certainement tronqué. Le fichier n'est pas analysé.
PNG : PNG Signature not found	La signature PNG n'est pas présente en début de fichier. Le fichier n'est pas analysé.
PNG : Unused data at the end of the file (N kb)	N kilobytes de données sont présentes après le bloc de fin de fichier (**IEND**).
PNG : Unused data (textfields and end of the file, total of N kb)	N kilobytes de données, réparties entre la fin du fichier et les blocs textuels, sont présentes au sein du fichier.
RS Analysis Positive detection, message Length : P	Le fichier est positif a l'analyse statistique RS. L'estimation de la longueur du message est donnée en pourcentage P de la capacité maximum.
WAV : Could not find Data bloc	Aucun bloc de données n'est défini dans le fichier WAV. Le fichier n'est pas analysé.
WAV : Could not find the Audio format Bloc	Aucun bloc de spécification de la nature des données audio n'est présent au sein du fichier. Le fichier n'est pas analysé.
WAV : Could not find the RIFF signature for the WAV file	La signature RIFF n'est pas présente en début de fichier. Le fichier n'est pas analysé.
WAV : File is not in WAV format, wrong FileFormatID	Le fichier n'est pas un fichier WAV. Il n'est pas analysé.
WAV : Truncated File	Le fichier est tronqué. Il n'est pas analysé.
WAV : Unused data at the end of the file (N kb)	N kilobytes de données inutilisées sont présentes en fin de fichier.

Message	Interprétation
WbStego signature detected : ciphertext sequential mode	La signature du logiciel WbStego a été trouvée dans le fichier. Il est déterminé que le fichier est crypté, et qu'il est dissimulé de manière séquentielle.
WbStego signature detected : distributed mode	La signature du logiciel WbStego a été trouvée dans le fichier. Il est déterminé que le fichier est dissimulé de manière distribuée le long du fichier.
WbStego signature detected : plaintext sequential mode (extension = N)	La signature du logiciel WbStego a été trouvée dans le fichier. Il est déterminé que le fichier est en clair, et qu'il est dissimulé de manière séquentielle. N en est l'extension.

11 Limitations connues

Les limitations de l'application concernent principalement le décodage des différents formats. En commençant par le BMP, seul les fichiers possédant une en-tête respectant le format Windows V3 sont décodés. Toutefois, ce format représente la très grande majorité des fichiers BMP actuels. Le format d'en-tête OS/2 est obsolète, et les versions plus récentes Windows (V4 et V5) ne sont pas utilisées pour des questions de compatibilité.

Le décodeur JPEG ne lit que les fichiers de type séquentiel. Les JPEG progressifs ne sont pas supportés, malgré leur forte utilisation sur Internet. Toutefois, il semblerait qu'il ne soit pas supporté par la majorité des logiciels de stéganographie. Les JPEG utilisant un codage arithmétique, ainsi que les JPEG avec compression sans perte, ne sont de même pas supportés.

L'application n'effectue pas de décompression des données PNG. Les données sont lues de manière brute. D'après l'analyse de marché effectuée précédemment, aucun des produits trouvés ne permet la dissimulation de données au sein du flux compressé.

Les fichiers tronqués, présents dans les données temporaires des navigateurs Internet ne sont pas traités. Il sont simplement placé dans le dossier des « Unprocessed files ».

Pour finir, lorsque la détermination du fichier est choisie par signature, un comportement anormal du EnScript fait que des fichiers non sélectionnés se retrouvent dans les résultats. Cela n'a pas d'incidence sur l'exécution ni même sur les résultats de l'application, ces quelques fichiers se retrouvent simplement dans les « Unprocessed files ».

12 Durée d'une analyse

La durée d'une analyse dépend, avant tout, des types de fichier qui sont à traiter. Le format nécessitant le plus de temps de traitement est le JPEG. Il se trouve, malheureusement, qu'il représente aussi un des formats les plus présents sur les machines. Pour donner un ordre de grandeur, un fichier JPEG de 863 Kb nécessite 30 secondes de traitement sur un ordinateur doté d'un processeur Pentium M à 1.86 GHz et 512 Mb de RAM. Par la prolifération d'appareils photo numérique, ce genre d'images est relativement fréquent.

Le tableau suivant montre le temps d'exécution de l'application selon plusieurs paramétrisation. Ces temps correspondent à l'exécution de l'application sur 1000 fichiers. Ces fichiers ont été pris au hasard, afin de permettre de s'approcher d'une analyse d'un cas réel, avec un nombre moins élevé de fichier.

Détermination du format		Analyses effectuées					-
Extension	Signature	JPEG	BMP	PNG	WAV	Générales	Temps
X		X	X	X	X	X	14 min
	X	X	X	X	X	X	14 min 6 s
X			X	X	X	X	17 s
	X		X	X	X	X	19 s

TAB. 7 – Temps d'exécution de l'application en fonction des options demandées.

Il n'est pas nécessaire d'effectuer plus de test, car il apparaît évident que les analyses JPEG représentent l'opération la plus coûteuse d'un point de vue temps d'exécution. La détermination du type de fichier par signature ne pénalise pas beaucoup les performances. Les quelques secondes supplémentairse représentent le temps d'initialisation du moteur de recherche des signatures (opération effectuée au début de l'exécution de l'application).

Ces chiffres sont uniquement à prendre comme ordre de grandeur. Au final, l'exécution du script est fortement dépendante de la nature des données analysées. Cela peut aller d'une heure, à plus d'un jour.

Troisième partie

Documentation Technique

13 Langage de programmation

L'application a été développée en utilisant le langage EnScript. Le EnScript est un langage de programmation propriétaire développé par GuidanceSoftware. Ce langage est interne au logiciel EnCase, ce dernier est donc requis pour l'exécution de l'application. Il s'agit d'un langage interprété, comme peut l'être le Perl.

La syntaxe du langage s'inspire des langages C++ et Java. Si la syntaxe est plus proche du C++, la méthodologie de programmation est similaire au Java. Tout comme ce dernier, une librairie standard conséquente est présente. La programmation se résume souvent à de longue recherche dans la documentation des classes.

L'utilité première du langage est de permettre l'automatisation des processus d'investigations numériques. La plupart des actions pouvant être effectuées de manière manuelle par le logiciel EnCase, possèdent leur pendant sous forme de classe EnScript.

Le EnScript est, comme les langages dont il s'inspire, orienté objets.

13.1 Faiblesses du langage

Une première limitation de ce langage est au niveau des performances. Comparativement au JAVA, les temps d'exécution sont extrêmement long. Il semble que le EnScript ait particulièrement de peine avec la gestion de tableau de grandes dimensions. L'application développée met ce manque d'efficacité particulièrement en évidence. Afin de pouvoir être analysé, les fichiers sont représentés sous forme de tableau de bytes. Cela représente une quantité non négligeable de données. Par exemple, une fichier BMP de 1280 sur 1024 pixels avec une profondeur de couleur de 24 bits, représente 3'932'160 bytes de données. Ce qui représente un tableau de la même taille.

Ce problème avec les grosses structures de données semble être dû à une gestion laborieuse de la mémoire. Ce phénomène est amplifié lorsque la structure en question est un tableau multidimensionnel. Malgré que cette possibilité soit offerte par le EnScript, l'utilisation en semble déconseillé. Tout d'abord, la déclaration ainsi que l'initialisation sont déconcertantes. Chaque type de tableau en EnScript nécessite d'effectuer une déclaration de type. La syntaxe en est la suivante :

```
typedef typeDeBase[] NomDuTypeTableau;
```

La définition d'un tableau bidimensionnel doit être effectuée en deux étapes. Commencer par définir le type de la première dimension, puis définir le type final en se basant sur celui précédemment crée. La syntaxe raccourcie suivante n'est pas possible :

```
typedef typeDeBase[][] NomDuTypeTableau;
```
[3]

Lors de l'initialisation de la taille du tableau, chacun des éléments doit être initialisé séparément. L'utilisation d'une structure de boucle est nécessaire.

[3] Cette syntaxe est correct (aucune erreur à la compilation). Cependant aucun moyen n'est offert permettant d'initialiser ce type de tableau.

```
typedef int[] IntArray;
typedef IntArray[] IntMatrix;

void Main(){
  IntMatrix m = new IntMatrix(2);
  foreach (IntArray a in m) a = new IntArray(3);
}
```

De plus, la libération de l'espace mémoire utilisé par cette structure prend beaucoup de temps. Cela peut aller jusqu'à plusieurs minutes et ne semble pas croître de manière linéaire. Il est donc préférable d'opter pour un tableau à une seul dimension, et de simuler la deuxième dimension par l'application d'offset aux indices du tableau.

Une seconde faiblesse est la pauvreté de sa librairie de fonctions mathématiques. Dans la première version de EnCase utilisée, des fonctions tels que la racine carrée ou le sinus d'un nombre n'étaient pas présentes. Depuis les mises à jour du produit ont permis d'étoffer un peu cela. Toutefois il manque toujours une fonction permettant la mise en puissance. Cela n'est pas très dérangeant dans le cadre d'une analyse de signature, il en est autrement lors de l'implémentation d'analyses statistiques.

La gestion des exceptions est aussi problématique. Aucun mécanisme similaire aux blocs `try...catch` en JAVA n'est disponible. Les fichiers analysés n'étant pas tous bien formés, ou même complets, il est nécessaire d'ajouter de nombreux tests. Chaque lecture dans le fichier, chaque accès aux données d'un tableau nécessite d'avoir préalablement vérifié que cette donnée existe. Lors de l'écriture d'un script en partant d'une feuille blanche, il est nécessaire de tenir compte de cela. Le plus difficile étant lors de l'adaptation d'un code existant d'un autre langage offrant une gestion poussée des exceptions (Java). Dans ce cas, l'entier de l'application est à repenser.

Les faiblesses énoncées ci-dessus ont été les plus gênantes lors du développement de l'application présentée dans ce document. D'autres faiblesses, notamment au niveau de l'initialisation de variables, peuvent être mentionnées. Cependant, ces dernières n'ont pas de réel impact sur le présent travail et ne sont donc pas détaillées.

14 Structure de l'application

Les classes composant l'application peuvent être regroupées par catégories. Au nombre de trois, ces catégories sont : Classes d'analyses, Classes de format de fichier et Classe d'interface utilisateur. Peut y être ajouté, la classe `Main` servant de moteur d'application.

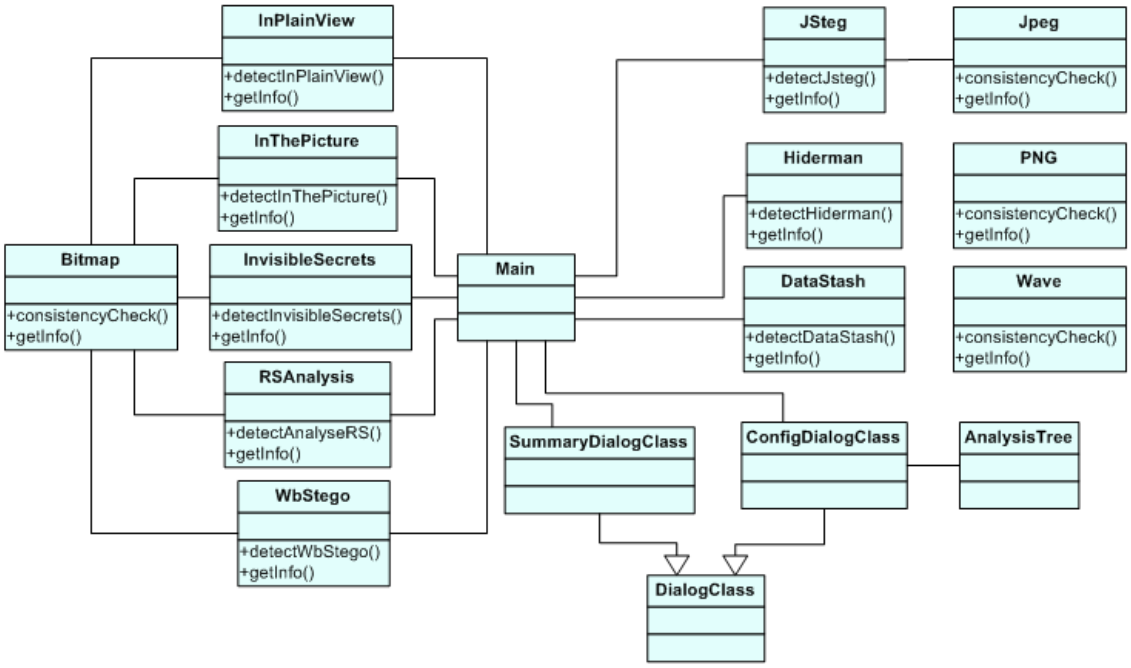

FIG. 25 – Diagramme UML (simplifié) de l'application.

14.1 Classes d'analyses

Les classes d'analyses représentent l'implémentation du travail effectué en section 4. Chacune de ces classes nécessite de contenir au minimum deux méthodes. Le reste de l'implémentation dépend du type d'analyse.

Méthode `bool detect`*`NomAnalyse`*`(TypeFichier fichier)`

Cette méthode constitue le coeur de l'analyse. Elle effectue l'analyse du fichier et retourne un booléen. Une valeur à `true` signifiant que le fichier passé en paramètre est positif à l'analyse en question. Le type du fichier passé en paramètre dépend du format auquel s'applique l'analyse. Dans l'état actuel de l'application, ils sont au nombre de trois : `Bitmap`, `Jpeg` et `EntryClass`. Cette dernière classe fait partie de la librairie standard EnScript. Elle est utilisée par les classes `Hiderman` et `DataStash`, ces dernières étant applicables à toutes sortes de formats de fichier.

L'un des point important à respecter lors de l'implémentation de cette fonction est de bien vérifier que suffisamment de données sont disponibles à l'analyse. L'oubli de ce point pourrait engendrer le plantage de l'application.

Méthode `String getInfo()`

Cette méthode est appelée lors d'une détection positive. Elle retourne des informations concernant la nature de la détection. Le niveau de détail dépend de l'analyse concernée. Dans certain cas, cela se limite à faire mention qu'une signature spécifique est présente. Dans d'autres, des informations complémentaires quant à la nature des données ou la manière de les dissimulés sont fournies. Cette méthode est utilisée afin de remplir le champ commentaire des objets de type `BookmarkClass`.

14.2 Classes de format de fichier

Ces classes implémentent le décodage d'un type particulier de format de fichier. Plusieurs objectifs leurs sont imposés. En premier lieu, elle vérifie que le fichier soit bien au bon format. Selon les options sélectionnées lors de la configuration de l'application, cette tâche est déjà effectuée par le moteur d'application. Toutefois, une deuxième vérification plus complète n'est pas inutile. Plusieurs variantes au sein du même format de fichier étant possible, un contrôle de la compatibilité avec le décodeur est mené. Il est de plus nécessaire d'implémenter ces classes de manière à pouvoir gérer la lecture de fichiers tronqués. Cette gestion consiste simplement à, dans un premier temps, détecter ce cas de figure. Puis de stopper le décodage du fichier et renseigner l'utilisateur de cette situation.

Ensuite, ces classes décomposent le fichier en plusieurs parties. En fonction du format, ces parties sont plus ou moins nombreuses.

Pour finir, leur dernier objectif est de fournir des méthodes permettant de vérifier la consistance du fichier. En se basant sur ce qui est énoncé en section 4.1, des tests sont implémentés afin de déterminer la présence de données non conformes et en estimer la quantité.

Attribut `bool ErrorOpening`

Cet attribut est utilisé pour informer le moteur d'application que le fichier n'a pas pu être correctement décodé. Une analyse de ce fichier n'est donc pas possible. Ce mécanisme est ajouté pour palier au manque de structure prédéfinie pour la gestion des exceptions en EnScript. Le moteur d'application vérifie la valeur de cet attribut. En cas d'erreur, le fichier est classé en tant que « Unprocessed file ».

Attribut `String errorMsg`

Cette chaîne de caractères fournit un supplément d'informations quant à la nature du problème rencontré. Elle peut être assimilée à la méthode `getInfo()`, mais dans le cadre de la gestion d'exceptions.

Les deux attributs précédents sont accédés de manière directe par `nomObjet.attribut`.

Méthode `bool consistencyCheck(int threshold)`

Cette méthode vérifie la consistance du fichier. Elle compare que la taille des données se trouvant à des emplacements inappropriés ne soit pas supérieure à un certain seuil. Ce dernier est défini par le paramètre `threshold`, donné en nombre de bytes.

Selon la nature du fichier, cette méthode fait appel à des sous méthodes spécifiques à chaque emplacement. Comme exemple, la classe `Jpeg` définit les deux sous méthodes suivantes :

```
bool consistancyCheckEnd(int threshold)
bool consistancyCheckComments(int threshold)
```

Méthode `String getInfo()`

Ces classes implémentant aussi des analyses - les analyses de consistance - , la méthode `getInfo()` y est aussi présente. Son objectif est le même que pour les classes d'analyses.

14.3 Classes d'interfaces utilisateurs

Ces classes définissent tout ce qui est en interaction avec l'utilisateur. Que se soit la saisie d'information, ou simplement l'affichage de résultats. Deux classes principales sont présentes : `ConfigDialogClass` et `SummaryDialogClass`. L'une comme l'autre héritent de la classe `DialogClass`. Elle permet la création d'une boîte de dialogue en EnScript et définit des méthodes permettant la gestion des interactions utilisateurs. Ces méthodes peuvent être surchargées par les classes enfants.

Méthode `virtual`[4] `void ChildEvent(const EventClass &event`[5]`)`

Cette méthode est appelée lorsque d'une interaction avec un élément de la boite de dialogue (bouton, zone de texte, etc...). Tous les évènements venant de cette dernière sont traités par cette méthode, la nature de l'évènement devra donc y être déterminé. Il est nécessaire de faire appel à l'implémentation de la classe parente.

Méthode `virtual bool CanClose()`

Comme son nom peut l'indiquer, cette méthode est appelée lors de la fermeture de la boite de dialogue. Elle défini les actions qui doivent être entreprises lors de cette fermeture. Tout comme la fonction ci-dessus, un appel de la méthode parente est nécessaire.

Classe `SummaryDialogClass`

Cette classe crée la boite de dialogue de résumé apparaissant à la fin de l'exécution des analyses. Elle consiste uniquement en un texte statique affichant les résultats de l'analyse. Le texte devant être affiché lui est fourni par le moteur d'application lors de la création de l'objet.

Classe `ConfigDialogClass`

Cette classe définit la boite de dialogue permettant la configuration des analyses à effectuer. L'élément central de cette interface est l'arbre représentant les analyses possibles. Il est affiché en utilisant un composant `TreeEditClass`. Il nécessite au préalable la définition d'un arbre en mémoire. Cette fonction est remplie par la classe `AnalysisTree`.

Au sein de cette classe, chaque catégorie d'analyse est représentée par un tableau de String. L'ajout d'une analyse à l'arbre s'effectue en insérant le nom de l'analyse dans le tableau de la catégorie impactée. Pour être complète, une modification de la méthode `CanClose()` de la classe `ConfigDialogClass` est nécessaire. L'implémentation de cette méthode est expliqué plus loin dans cette section.

[4] Le mot clé `virtual` mentionne que la fonction peut être surchargée.

[5] Le `&` permet de spécifier que le paramètre est passé par référence. Le mot clé `const` definit que le paramètre ne peut pas être modifié.

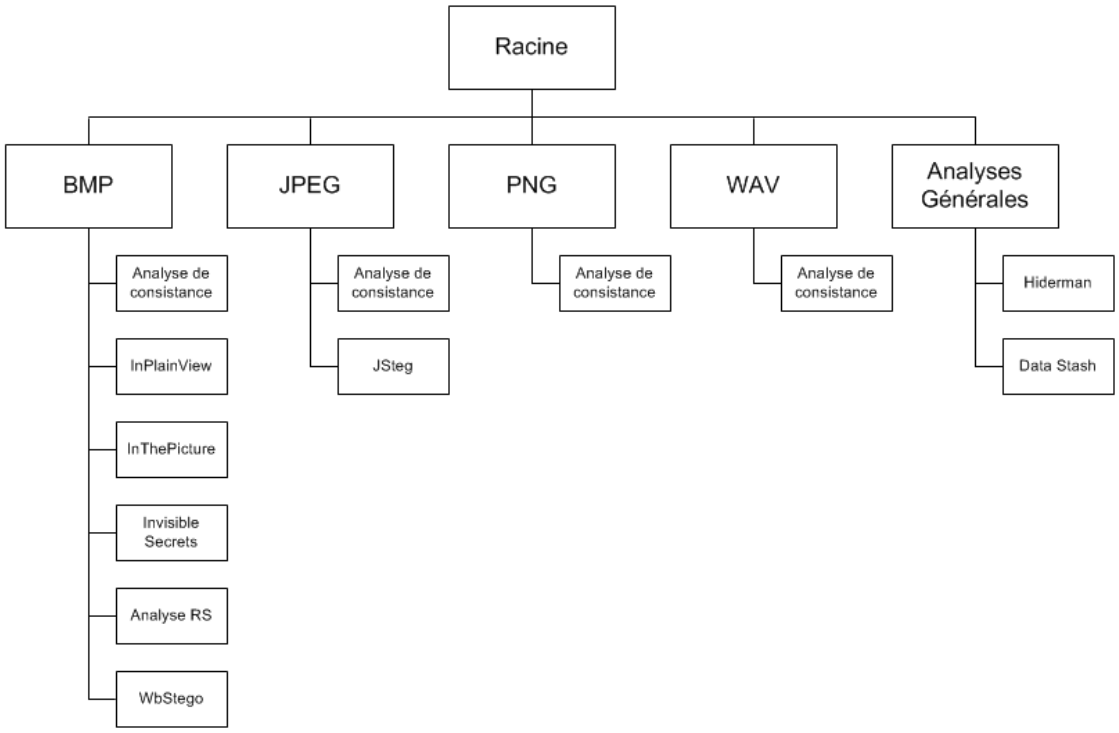

FIG. 26 – Arbre des analyses.

Cette arborescence est définie à l'aide d'objets de type NameListClass. Cette classe permet de créer une liste d'objet de type String. Un arbre peut simplement être crée à l'aide du code suivant :

```
// Declaration de la racine de l'arbre
tree = new NameListClass(null,"Analysis");

NameListClass bmpTree(tree, "BMP");
for(int i=0; i < bmpAnalysis.Count();i++){
   new NameListClass(bmpTree, bmpAnalysis[i]);
}
```

Dans cette exemple, une sous catégorie « BMP » est définie. Puis tous les objets présents dans le tableau `bmpAnalysis` y sont ajouter. Cette étape est renouvelée pour chacun des formats pris en charge.

La fonction la plus important de cette classe est la fonction surchargée CanClose(). Sa fonction est de parcourir l'arbre des analyses, à la recherche des éléments sélectionnés. En fonction de ceux-ci, les valeurs booléens permettant de définir si une analyse doit être effectuée sont modifiées. Le moteur d'application vient ensuite lire ses valeurs afin de créer l'environnement d'analyse correspondant au souhait de l'utilisateur.

14.4 Moteur d'application

Comme dit précédemment, le moteur d'application est implémenté dans la classe `Main` directement. Les tâches qu'elle doit gérer sont diverses. C'est elle qui est chargée de la

création des objets décrits précédemment, ainsi que du séquençage des opérations. Son fonctionnement global est décrit par la figure 27.

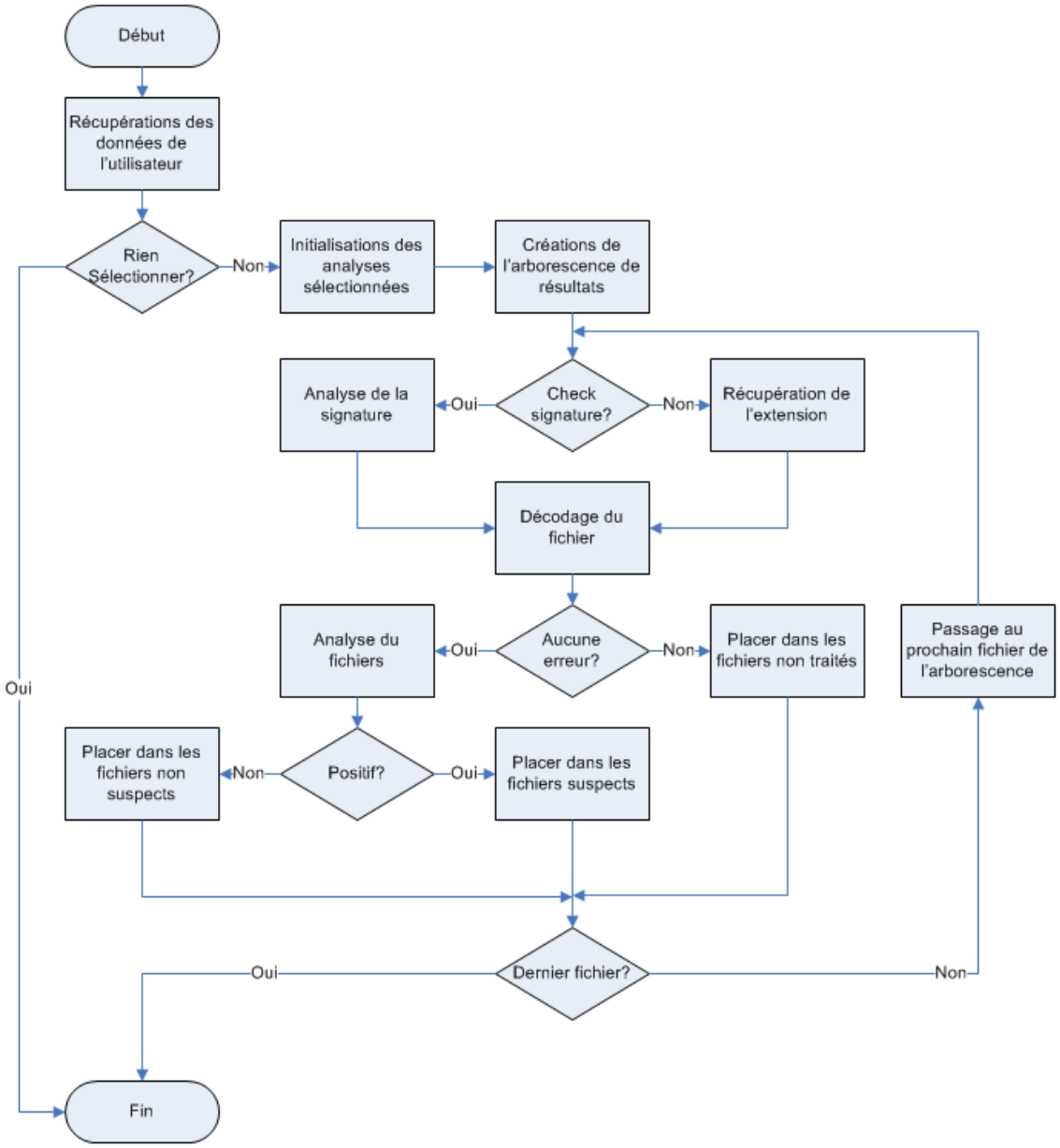

FIG. 27 – Diagramme de flux du moteur d'application.

Elle commence donc par récupérer les données saisies par l'utilisateur par l'intermédiaire de la classe `ConfigDialogClass`. Une vérification est effectuée dans le cas où aucune analyse n'aurait été sélectionnée. Dans ce cas, l'exécution de l'application s'arrête simplement. Il suit une création conditionnel des objets d'analyses.

L'arborescence des résultats est crée en utilisant des objets `BookmarksFolderClass`. Comme son nom l'indique, cette classe permet de définir des dossiers dans la vue Bookmarks. Ces objets sont utilisés lors de l'ajout d'un fichier dans une des catégories définies. L'arborescence est définie de la manière suivante :

```
BookmarkFolderClass folder(c.BookmarkRoot(), cdc.bookFolderName);
BookmarkFolderClass suspicious(folder, "Suspicious Content");
BookmarkFolderClass notsuspicious(folder, "Apparently not suspicious content");
BookmarkFolderClass unproc(folder, "unprocessed file(s)");
```

`c.BookmarkRoot()` correspond au dossier racine des Bookmarks, `cdc.bookFolderName` est le nom du dossier que l'utilisateur a dû entrer lors de la configuration du script.

Après cette étape, la partie créant l'environnement de l'application est terminée et l'analyse des fichiers à proprement parler commence. Cela débute par la vérification de la signature, si elle est demandée. Cette étape se base sur la comparaison de la signature du fichier traité, avec les éléments de la base présente dans EnCase. Le fonctionnement en est le suivant. La vérification de signature intégrée à EnCase regarde si il y a une correspondance entre l'extension du fichier et la signature. Si tel est le cas, « Match » est renvoyé mentionnant la correspondance. L'extension peut être utilisée pour déterminer le format du fichier. Lorsque la signature ne correspond pas à l'extension, plusieurs cas de figure sont possibles. Si cette signature est présente dans la base EnCase, le nom de cette signature précédé du caractère * est renvoyé. Par exemple, dans le cadre d'un fichier JPEG, le résultat est égal à « * JPEG Image Standard ». Si l'extension est présente dans la base, mais que la signature ne correspond à aucun élément, le résultat vaut «! Bad signature ». Enfin, si ni la signature ni l'extension ne sont présents dans la base, le fichier est défini comme « Unknown ».

Lorsque le type de fichier est identifié, le fichier peut être analysé. Un bloc conditionnel est défini pour chacun des formats pris en charge par l'application. Le squelette de ces bloc est le suivant :

```
if (fileFormat.Compare(extensionFichier)==0){
   bool process = cond; // condition pour traiter le fichier

   TypeDeFichier fichier = process ? new TypeDeFichier(entry) : null;

   if (fichier != null && img.ErrorOpening){
      unproc.AddBookmark(entry,0,0,fichier.errorMsg,bookOptions,bookView);
      img = null;
      continue;
   }

   // Liste des analyses pouvant etre appliquee au format

   img = null;
}
```

La variable `process` spécifie si le fichier doit être traité par le décodeur. L'utilité étant de ne pas décoder le fichier si aucune analyse sélectionnée n'y si applique. Le fichier est ensuite décodé. Après cette opération il est vérifié que la lecture s'est correctement déroulée. Si tel n'est pas le cas, le fichier courant est ajouté au dossier des fichiers non traités et l'application passe au fichier suivant. Autrement, selon la demande de l'utilisateur, les analyses demandés sont effectuées sur le fichier.

Lorsque tous les fichiers ont été parcourus, le temps d'exécution de l'analyse est comptabilisé ainsi que le nombre de détection par type d'analyse. Ces informations sont affichées dans la fenêtre de résumé qui est affichée en fin d'exécution.

15 Détection de programmes stéganographiques

EnCase offre de manière native la possibilité de parcourir une arborescence de fichiers en comparant les hashs MD5 des fichiers présents avec une base de hash. Ces bases sont appelées des Hash sets. Elles permettent de définir une liste de fichiers potentiellement suspects, afin de détecter leur présence sur l'ordinateur analysé.

Lors de précédent travail, une base contenant les hashs de quelques logiciels stéganographique avait été créée. Pour débuter, ces hashs sont récupérés. Malgré que de nombreux programmes présents dans cette base soit relativement vieux, il est utile de les laisser. Personne ne peut en effet affirmer que ces logiciels ne sont plus utilisés. Ensuite, de nouveaux hashs sont ajoutés.

Afin de pouvoir être importée dans EnCase, la base doit respecter le format Hashkeeper. Ce format requière la création de deux fichiers distincts : un fichier .hsh et un fichier .hke portant le même nom de fichier. Le fichier .hke définit la nature du Hash Set, alors que le .hsh contient les hashs à proprement parler. Le fichier .hsh, pour commencer, doit avoir le format suivant :

```
"file_id","hashset_id","file_name","directory","hash","file_size",
"date_modified","time_modified","time_zone","comments","date_accessed",
"time_accessed"
```

Le tout séparé uniquement par des virgules. La première ligne du fichier doit comporter la description ci-dessus. Chaque hash est séparé par un retour à la ligne. Il n'est pas nécessaire que tous les champs spécifiés dans le format soient remplis. Le champ peut simplement être laissé vide, toutefois les virgules devront tout de même être présentes. Pour cette application, seul les champs `file_id`, `hashset_id`, `Hash` et `comments` sont renseignés. Voici un exemple du contenu du fichier :

```
1,1,,,"d451da2cc992481f474d4b22073e17d0",,,,,"CryptoBola JPEG trial",,
2,1,,,"e566925498d22aa7cc54ee358d4bf0e8",,,,,"Xidie executable",,
3,1,,,"8ed334c8f73b37e81df210b97c4a51d3",,,,,"Stegowav archive",,
4,1,,,"85b152b53b96cbedfe5ce8b42878a8a7",,,,,"Xidie installeur",,
```

Le champ `file_id` doit simplement être incrémenté à chaque nouvelle valeur de Hash, alors que le champ `hashset_id` doit correspondre à une valeur présente dans le fichier .hke. Le format de ce second fichier ressemble beaucoup au premier :

```
"hashset_id","name","vendor","package","version"," authenticated_flag",
"notable_flag","initials","num _of_files","description","date_loaded"
```

Tout comme le premier fichier, il n'est pas nécessaire de remplir tous les champs. Cette définition des champs doit aussi être présente en première ligne du fichier. Un seul Hash Set étant nécessaire, ce fichier ne contient que l'entrée suivante :

```
1,"Steganographic tools",„,1,1,,0,"Hashes of some steganographic tools",
```

Le format qui doit être respecté étant connu, le format du fichier provenant de Steganographia doit être modifié. Cette tâche à été automatisée par l'écriture d'un petit utilitaire en JAVA.

16 BMP

Le format BMP est un format d'image introduit par Microsoft et IBM. Il est aussi connu sous le nom de DIB (device-independent bitmap). Sa large distribution provient de son utilisation de manière native dans les Systèmes d'exploitation Microsoft Windows et IBM OS/2.

La structure d'un fichier BMP est définie de manière très simple. Il représente le format de fichier image le plus facile à développer. Cela explique qu'il soit le format le plus représenté dans les logiciels de stéganographie (voir la section 3.2 en page 4). Malgré qu'il offre la possibilité de compresser les données, le format BMP est principalement vu comme un format d'image sans compression.

Un fichier BMP est formé de 4 parties distinctes : l'en-tête de fichier (File Header), l'en-tête Bitmap (Bitmap Information Header), la palette de couleurs et pour finir les données de l'image. Ayant été développé initialement pour PC à base de processeurs Intel, la représentation des valeurs se conforme au codage *little-endian*[6].

16.1 File Header

Cette en-tête contient les information concernant le fichier de manière générale. Deux champs sont importants au niveau de cette en-tête : le champ `bfType` et le champ `bfOffBits`.

Le champ `bfType` contient la signature du fichier BMP. Ce dernier contient toujours les deux bytes `0x42` `0x4D` (caractère ASCII B et M). Le deuxième champ, quant à lui, définit l'adresse de départ des données de l'image relativement au début du fichier.

Champ	Description	Taille [Byte]
bfType	Signature Bitmap	2
bfSize	Taille du fichier Bitmap [Byte]	4
bfReserved1	Champ reservé	2
bfReserved2	Champ reservé	2
bfOffBits	Offset de l'image	4

TAB. 8 – Format du File Header.

16.2 Bitmap Info Header

Le Bitmap Info Header contient les informations nécessaires au décodage de l'image. Plusieurs versions de cette en-tête existent, apportées par chaque nouvelle version de l'OS de

[6]Dans ce codage, les **bytes** de poids faible précédent ceux de poids fort. Un nombre donné par la valeur `0x3C62` sera écrit sur le fichier `0x623C`.

microsoft. Cependant, pour des raisons de compatibilité, les encodeurs actuels continuent d'utiliser la version Windows V3. Les versions plus récentes, ainsi que les versions OS/2 de cette en-tête ne sont pas vraiment utilisées.

Champ	Description	Taille [Byte]
biSize	Taille de l'en-tête (40 bytes)	4
biWidth	Largeur de l'image en pixels	4
biHeight	Hauteur de l'image en pixels	4
biPlanes	Nombre de plan de bit	2
biBitCount	Nombre de bits par pixel	2
biCompression	Methode de compression	4
biSizeImage	Taille du fichier	4
biXPelsPerMeter	La résolution horizontale (pixel par mètre)	4
biXPelsPerMeter	La résolution verticale (pixel par mètre)	4
biClrUsed	Nombre de couleur dans la palette	4
biClrImportant	Nombre de couleur importante	4

TAB. 9 – Format du Bitmap Header (Windows V3).

Cette en-tête est d'une longueur fixe de 40 bytes. Les champs `biWidth` et `biHeight` contiennent des entiers signés. Une hauteur négative signifie que le fichier est encodé en partant de haut en bas (l'encodage standard étant de bas en haut, de gauche à droite).

16.3 Palette de couleurs

Une palette est définie pour les images constituées de 1,4 ou 8 bits par pixel (non utilisé pour les images en niveau de gris). Elle suit directement le Bitmap Info Header. Le nombre de couleurs contenues dans la palette peut être calculé par la formule suivante :

$$Nb_{color} = \frac{Offset - Size_{FileHeader} - Size_{InfoHeader}}{4}$$

Chaque couleur est représentée par la valeur des composantes RGB, suivie par un byte à 0 n'étant pas utilisé. La composante RGB est écrite en commençant par la fin (donc blue, green et red).

16.4 Données de l'image

L'image est représentée sous la forme d'une matrice de pixels. Comme mentionné plus haut, le codage standard de l'image s'effectue en partant du coin inférieur gauche. Chaque ligne doit nécessairement être un multiple de 4 bytes. Si cela n'est pas le cas, des bytes de padding sont ajoutés en fin de ligne.

La représentation des pixels dépend de la profondeur des couleurs choisie. Cela va de 8 pixels par byte (1 bit par pixel) à 4 bytes par pixel (32 bits). Toutefois, ces deux extrêmes ne sont pas très répandus. 8 et 24 bits par pixel sont les représentations les plus communes.

Dans le cas d'une image couleur de 8 bits, la couleur du pixel est représentée par un indice dans la palette définie. Pour le cas d'une image 24 bits, chaque composante couleur est définie à l'aide d'un byte. L'ordre dans lequel ils apparaissent est le même que celui utilisé pour la palette de couleurs.

17 JPEG

La norme JPEG étant complexe, elle n'est pas détaillé dans le présent document. Une description général peut être trouvée en Annexe D. Pour plus de détails, se référer au document [25].

18 PNG

Le format PNG a été développé avec pour objectif de remplacer le format GIF, ce dernier étant devenu propriétaire du faite de l'utilisation de l'algorithme LZW faisant l'objet d'un brevet déposer par UniSys [21]. Cependant, le PNG n'est pas uniquement un dérivé du format GIF utilisant un algorithme de compression libre. Il efface les quelques limitations indésirables imposées par ce dernier (limitation du nombre de couleurs à 256).

Contrairement au JPEG, l'algorithme de compression utilisé est sans perte. Il s'agit de l'algorithme DEFLATE utilisé par la librairie zlib. Cette algorithme, libre de droit, fut développer à l'origine pour une utilisation par le format de fichier ZIP. C'est une combinaison entre l'algorithme LZ77 (précurseur du LZW breveter par Unisys) et un codage de Huffman. Il est spécifié dans la RFC 1951 [22].

La représentation des nombres respecte l'encodage *big-endian* tout comme le JPEG/JFIF. Le fichier commence par une signature de 8 Bytes donnée en hexadécimal par :

```
89 50 4E 47 0D 0A 1A 0A
```

A l'exception de cette signature, les données sont regroupées par bloc appelé *Chunks*. Ils regroupe les données de même nature. Dans le tableau 10 est spécifié le format que doit respecter chacun de ces blocs, indépendamment de leur nature. Les champs sont donnés dans l'ordre dans lequel il doivent apparaître dans le fichier.

Nom	Taille [Byte]	Description
Length	4	Longueur de la partie "Chunk data"
Chunk type	4	Définition du type de bloc (identifié par 4 caractère ASCII)
Chunk data	-	Les données du bloc
CRC	4	Checksum

Tab. 10 – Format d'un bloc PNG.

Comme le montre le tableau précédent, chacun de ces blocs est protégé par un CRC. Il est calculé sur les champs « Chunk type » et « Chunk data ». Il est obligatoire pour tous les blocs, même ceux ne contenant aucune donnée.

Les spécifications du format PNG définissent plusieurs types de blocs. Ils peuvent être divisés en deux catégories : Les blocs obligatoires (*Critical chunks*) et les blocs optionnels (*Ancillary chunks*). Les blocs obligatoires sont nécessaires au décodage du fichier. Un fichier n'est pas considéré comme valide si un de ces champs vient à manquer. Ils sont au nombre de 4 : **IHDR**, **PLTE**, **IDAT** et **IEND**.

18.1 Bloc IHDR

Ce bloc contient l'en-tête de l'image. Son type est donné par la valeur `0x49484452`. Il doit obligatoirement suivre la signature. Ci-dessous, un tableau représentant les champs présents dans l'en-tête.

Nom	Taille [Byte]
Width	4
Height	4
Bit depth	1
Colour type	1
Compression method	1
Filter method	1
Interlace method	1

TAB. 11 – Format de l'en-tête PNG.

Le Bloc **IHDR** est de taille fixe. La longueur du bloc sera donc toujours de 13 bytes. Toutes les valeurs de l'en-tête sont représentées par des entiers non signés.

Remarque Dans la spécification actuelle du format, seul la valeur 0 est définie pour le champ « Compression method ». Elle correspond à une compression par l'algorithme DEFLATE. Les autres valeurs sont réservées à de futures extensions.

18.2 Bloc PLTE

Malgré que ce bloc fasse partie des blocs dits obligatoires, il n'est pas obligatoirement présent dans chaque fichier.Il est défini par la valeur `0x504C5445` pour le champ type et représente la palette de couleurs. Ce bloc n'est obligatoire que dans un cas particuliers (colour type = 3, couleurs indexées), mais peut-être optionnel dans d'autres cas.

Les données peuvent représenter 1 à 256 couleurs, chacune d'entre elles étant représentées par 1 Byte par composante RGB dans l'ordre Red, Green puis Blue. Le nombre de couleurs défini dans la palette peut être calculé à l'aide du champ length du bloc ($\frac{length}{3}$).

18.3 Bloc IDAT

Ces types de blocs, représentés par la valeur `0x49444154`, contiennent les données de l'image. Les données sont représentées comme un flux de données compressées par l'algorithme DEFLATE. Les données peuvent être réparties dans plusieurs blocs **IDAT**. Si tel est le cas, ces blocs doivent nécessairement être consécutifs. Dans ce cas particulier, les données compressées sont obtenues par concaténation du contenu des blocs.

18.4 Bloc IEND

Ce bloc définit la fin de l'image. La valeur du champ type est donnée par l'entier sur 32 bits `0x49454E44` sous forme hexadécimale. Aucune donnée ne doit être présente après ce bloc. Comme tous les autres blocs, il est composé d'un champs spécifiant la longueur des données, ainsi que d'un CRC. La longueur des données est toujours à 0, car ce bloc ne comporte pas de données.

18.5 Blocs Optionnels

La spécification PNG définit de nombreux blocs optionnels. Dans cette partie, vont être détaillés ceux représentant un intérêt d'un point de vue de dissimulation d'informations.

Les trois blocs définis comme intéressants ont un point commun, leur but est l'ajout d'informations textuelles à l'image. Il sont représenté par les type **tEXt**, **zTXt** et **iTXt**.

Le bloc **tEXt** est le plus basique. Son type est donné par la valeur `0x74455874`. Il permet la saisie de couples mot clé texte, en se limitant au jeu de caractère ISO8859-1 (Latin-1). Les données saisies ne peuvent pas comporter de caractère null (byte 0), ce qui limite ce type de blocs uniquement au texte.

Description	Taille [byte]
Mot clé	1-79
Séparateur	1 (caractère null)
Texte	≥ 0

Tab. 12 – Format du bloc « tEXt ».

Les blocs **zTXt**, dont le champ type vaut `0x74545874`, sont identiques au précédent. Ils diffèrent uniquement par le fait les données contenues sont compressées. L'algorithme de compression utilisé est le même que pour les images (DEFLATE).

Description	Taille [byte]
Mot clé	1-79
Séparateur	1 (caractère null)
Compression	1
Texte compressé	N

Tab. 13 – Format du bloc « zTXt ».

Pour finir, le bloc **iTXt** permet de saisir du texte, mais en utilisant le jeu de caractère UTF-8. La valeur de son champ type est donné par `0x69545874`. Tout comme le bloc **tEXt**, les

données ne peuvent pas contenir de caractère null (byte 0). Ce bloc peut optionnellement être compressé par DEFLATE.

Description	Taille [byte]
Mot clé	1-79
Séparateur	1 (caractère null)
Compression « flag »	1
Compression	1
Language « tag »	≥0
Séparateur	1 (caractère null)
Mot clé traduit	≥0
Séparateur	1 (caractère null)
Texte	≥0

Tab. 14 – Format du bloc « iTXt ».

Chacun de ces blocs présente le désavantage de ne pouvoir contenir que des données textuelles. Cette faiblesse n'est cependant réelle que si ces données sont effectivement lues par les décodeurs. Ceci n'est pas majoritairement le cas.

19 WAV

Le WAV est un standard défini par Microsoft et IBM pour le stockage de données audio numériques. Il ne spécifie pas une méthode de codage ou de compression de l'information. C'est uniquement un conteneur se basant sur d'autres standards pour le codage des données audio. Cependant, la grande majorité des applications utilisent le codage PCM (*Pulse Coded Modulation)*, correspondant à des données brutes non compressées.

Le format se base sur les spécifications RIFF définissant le stockage des données par l'utilisation de Bloc de données (*Chunks*). Le format ayant été développé pour Windows, donc sur processeur Intel, les valeurs sont stockées en utilisant l'ordre *little-Endian*.

19.1 Organisation du fichier

Comme mentionné précédemment, le fichier est organisé en bloc. Plusieurs type de blocs sont définis. Chacun ayant sa propre utilité. Chaque bloc est composé de trois parties distinctes : un ID, la taille du bloc et les données du bloc. La taille du bloc spécifie uniquement la taille de la partie données, de manière à facilement pouvoir sauter un bloc non nécessaire.

Une spécificité du format RIFF, et donc du WAV, est que les données doivent être alignées au mot (2 Bytes). La taille totale doit donc nécessairement être paire. Dans le cas contraire, un byte de *padding* (de valeur 0) est ajouté à la suite des données. Il n'est pas pris en compte dans la longueur du bloc spécifié dans l'en-tête.

Un fichier WAV est principalement constitué de trois blocs : « RIFF », « fmt » et « data ». Il y a une hiérarchie entre les blocs. Les blocs « fmt » et « data » sont des sous blocs de « RIFF » (voir figure 28).

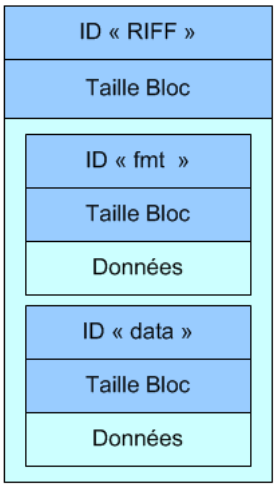

FIG. 28 – Structure globale d'un fichier WAV.

19.2 Bloc « RIFF »

Le bloc « RIFF » est le parent de tout autre bloc. Il ne possède pas de données propres, mais contient les autres blocs. En voici le format :

nom	taille [byte]	valeur
Chunk ID	4	0x52494646 (« RIFF »)
Chunk Data Size	4	Taille fichier - 8
RIFF Type	4	0x57415645 (« WAVE »)
-	-	Sous Blocs

TAB. 15 – Format du bloc « RIFF ».

Le RIFF étant un format général utilisé par plusieurs autres standards (notamment AVI), le champ RIFF Type permet de spécifier le type de sous blocs présents.

Remarque La lecture des champs contenant des chaînes de caractères doit se faire en utilisant l'ordre *Big-Endian*. Par exemple, pour les valeurs Chunk ID et RIFF Type.

19.3 Blocs WAV

Le standard WAV spécifie un grand nombre de type de bloc. La plupart d'entre eux sont optionnels et ne sont que très peu utilisés par les encodeurs WAV. La plupart des fichiers ne contiennent que deux blocs distincts : le bloc « fmt » et le bloc « data ».

19.3.1 Bloc « fmt »

Ce bloc définit la manière dont doit être interprété la bloc de données. Il correspond à l'en-tête du fichier WAV. Malgré que les spécifications ne fixent pas l'ordre des blocs dans

le fichier, il est conseillé de placer ce bloc avant le bloc de données (permet de décoder les données à la volée). Cette recommandation est respectée par la majorité des encodeurs.

nom	taille [Byte]	valeur
Chunk ID	4	0x666D7420 (« fmt »)
Chunk Data Size	4	16+
Compression code	2	1 - 65535
Number of channels	2	1 - 65535
Sample rate	4	1 - 0xFFFFFFFF
Average bytes per second	4	1 - 0xFFFFFFFF
Block align	2	1 - 65535
Significant bits per sample	2	1 - 65535
Extra format bytes	2	0 - 65535
-		Extra format bytes

Tab. 16 – Format du bloc « fmt ».

La taille du bloc dépend du nombre de bytes optionnels définis. 16 bytes sont requis, ensuite cela dépend de la compression choisie pour les données. Dans le cas d'un fichier WAV/PCM, aucun byte supplémentaire n'est ajouté.

19.3.2 Bloc « data »

Le bloc « data » contient les échantillons numériques du fichier audio. Le codage de l'information dépend de la méthode de compression utilisée. Voici le format général du bloc :

nom	taille	valeur
Chunk ID	4	0x64617461 (« data »)
Chunk Data Size	4	dépendant du contenu
-	Chunk Size	Échantillons

Tab. 17 – Format du bloc « data ».

Dans le cadre d'un fichier codé en PCM, les données sont stockées de manière entrelacée. Si le fichier contient plusieurs canaux, chaque échantillon de la même base de temps est écrit pour chaque canaux avant de passer au suivant. La figure 29 donne un exemple pour un fichier PCM stéréo (2 canaux), de 8 bits par échantillon.

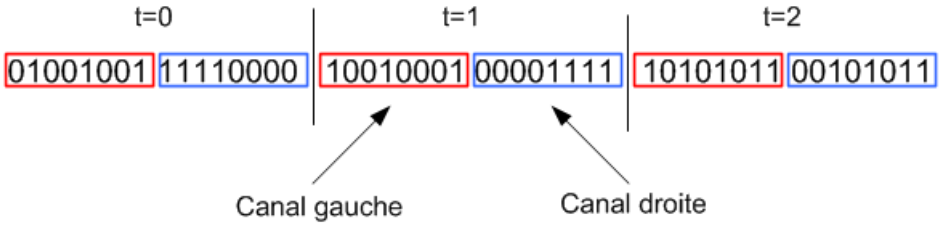

FIG. 29 – Codage PCM dans un fichier WAV.

Le même principe est appliqué pour le fichier audio de plus de 2 canaux. L'utilité étant que la lecture du fichier peut débuter avant d'avoir complètement transféré les données (streaming).

Références

[1] Niels Provos et Peter Honeyman, (2003).
Hide and Seek : An Introduction to Steganography.
IEEE Computer Society.

[2] Audio Steg.
`http ://www.snotmonkey.com/work/school/405/overview.html`

[3] Andreas Westfeld et Andreas Pfitzmann.
Attacks on Steganographic Systems.
Desden University of Technology, Germany.

[4] Mehdi Kharrazi, Husrev T.Sencar et Nasir Memon, (2004).
Image Steganography : Concepts and Practice.
Polytechnic University, Brooklyn.

[5] W.Bender, D.Gruhl, N.Morimoto et A.Lu, (1996).
Techniques for data hiding.
IBM SYSTEMS JOURNAL, VOL 35, NOS 3&4

[6] Pierre Le Chapelain, (2003).
Analyse steganographique d'images numeriques. Comparaison de differentes methodes.
Université de Joseph Fourier, France.

[7] Jean-Paul Delahaye, (1996)
Information noyée, information cachée.
Pour la science, N°229 Novembre 1996.

[8] Compression JPEG.
`http ://fr.wikipedia.org/wiki/Compression_JPEG`

[9] Jessica Fridrich, Miroslav Goljan et Dorin Hogea, (2000).
Attacking the Outguess.
Departement of Electrical and Computer Engineering, SUNY Binghamton.

[10] Niels Provos, (2001).
Probabilistic Methods for Improving Information Hiding.
CITI Technical Report 01-1.

[11] Niels Provos et Peter Honeyman.
Detecting Steganographic Content on the Internet.
Center for Information Technology Integration, University of Michigan.

[12] Neil F. Johnson et Sushil Jajodia, (1998).
Steganalysis : The Investigation of Hidden Information.
IEEE Information Technology Conference.

[13] Eric Cole, (2003).
Hiding in Plain Sight : Steganography and the Art of Covert Communication.
ISBN : 0-471-44449-9

[14] Jessica Fridrich, Miroslav Goljan et Dorin Hogea.
Steganalysis of JPEG Images : Breaking the F5 Algorithm.
Departement of Electrical and Computer Engineering, SUNY Binghamton.

[15] Andrew D. Ker, (2005).
Resampling and the Detection of LSB Matching in Bitmaps.
Oxford University Computing Laboratory, England.

[16] Analyse de programme stéganographique.
`http ://www.guillermito2.net/stegano/index.html`

[17] Jessica Fridrich, Miroslav Goljan et Rui Du, (2001).
Detecting LSB Steganography in Color and Gray-Scale Images.
IEEE Journal October-December 2001

[18] Transformée en cosinus discrète.
`http ://fr.wikipedia.org/wiki/Transform%C3%A9e_en_cosinus_discr%C3%A8te`

[19] Chen Ming, Zhang Ru, Niu Xinxin et Yang Yixian, (2006).
Analysis of Current Steganography Tools : Classifications & Features.
Information Security Center, Beijing University of Posts & Telecomm.

[20] Eiji Kawaguchi et Richard O. Eason.
Principle and applications of BPCS-Steganography.
University of Maine, Orono.

[21] The GIF controversy.
`http ://en.wikipedia.org/wiki/Graphics_Interchange_Format`

[22] RFC 1951 (DEFLATE Compressed Data Format).
`http ://tools.ietf.org/html/rfc1951`

[23] Micheal T.Raggo, (2000).
Steganography, Steganalysis & Cryptanalysis.
Black Hat Conference.

[24] Wave File Format.
`http ://www.sonicspot.com/guide/wavefiles.html`

[25] International Telecommunication Union, (1992).
Information Technology – Digital compression and coding of continuous-tone still images – Requirements and Guidelines.
Recommendation T.81.

[26] Guidance Software, (2005).
EnCase EnScript Language Reference Version 5.

[27] Guidance Software Message Board.
`https ://support.guidancesoftware.com/`

Quatrième partie

Annexes

A Logiciels contenus dans la base de Hash

Nom	Version	Fichier détecté
Blindside	0.9	BSIDE.EXE
BmpSteg	1.0	bmpSteg.jar
Cameleon	1.0	Cameleon.exe cameleon.zip cameleon.exe
Camouflage	1.2.1	Camouflage.exe CamShell.dll Camou121.exe
CryptArkan	1.0f	CryptArkan.exe bmp24b1b.dll bmp24b2b.dll bmp24b4b.dll wav16s1.dll wav16s2.dll wav16s4.dll wav16s8.dll CryptArkanSetup.exe
CryptoBola JPEG	2.1.0.1	CB_JPEG_EC_000.EXE
Data Privacy tools	7.5	dpt35.exe
Data Stash	1.5	DataStash.exe datastash.hlp ds.msi ds_sw.zip
Encrypt Pic	1.3	EncrPic.exe encpic13.exe
F5	release 12	Embed.class Extract.class ms_e.bat ms_d.bat Embed.java Extract.java
FortKnox	3.56	FortKnox.exe FortKnox.CAB

Hermetic Stego	7.41	hst.dll hst741.exe hst_setup.exe hst_setup.zip
Hide Password in Picture Encryptor	1.0	HidePassword.exe HidePassword.CAB
Hiderman	3.0	Aide.exe Hiderman.exe Hiderman_us.exe uninstall.exe
HideSeek	5.0	HIDESEEK.EXE hdsk50.zip
Hide4PGP	1.0	HIDE4PGP.EXE hide4pgp.zip
	2.0	H4PGP20W.ZIP hide4pgp.exe
ImageHide	2.0	ImageHide.exe
InPlainView	1.0	InPlainView.exe
InThePicture	2.2	ITP.exe ITP220.zip
Invisible Secrets	2.1	invsecr2.exe isecrets2.exe ISECRETS2.HLP uninstall.exe
	4.0	invsecr.exe invtray.exe
JPegX	1.00.6	jpgx.exe
JSteg Shell	2.0	JStegShell20.exe jsteg.zip
Puff	1.01	Puffv101.exe
SecurEngine Professional	1.0	SecurEnginePro.exe SEPBmpCarrier.dll SEPGifCarrier.dll SEPHtmlCarrier.dll SEPPngCarrier.dll securengine_setup.exe
S-Tools	4.0	S-Tools.exe
Steghide	0.5.1	steghide.exe

Steganos Security Suite	7.1.6	SSS7.dll SSS7.DLL SSSDE_Test.chm SSSEN_Test.chm SSSFR_Test.chm SSSJP_Test.chm SSSPL_Test.chm SSS7op.dll SSS7se.dll SSS7setip.dll suite.dll Steganos Security Suite 7.1.6.exe
Steganos Privacy Suite 2008	10.0.7	FileManager.exe Suite.exe sss2008int.exe wxbase28uh_net_vc.dll wxbase28uh_vc.dll wxbase28uh_xml_vc.dll wxmsw28uh_adv_vc.dll wxmsw28uh_core_vc.dll wxmsw28uh_gizmos_vc.dll wxmsw28uh_html_vc.dll
Stella	1.0	stella.jar
WbStego4Open	4.3	wbs43en.hlp wbStego4.3open.exe
Xidie	2.0	SteganoTools.dll Xidie.exe XidieSetup.msi XidieSetup.zip

B Logiciels de stéganographie cités dans ce document

Donne une liste exhaustive des logiciels de stéganographie cité au court de ce travail.

Absolute Password Protector LastBit Software
Version : 1.0
Licence : Shareware, 29 €
Plateformes : Windows
Site : `http ://lastbit.com/app/`

Cryptix Rbcafe.com
Version : 0.64b
Licence : Commercial, 12 €
Plateformes : Mac
Site : `http ://www.rbcafe.com/cryptix`

Data Stash Skyjuice Software
Version : 1.5
Licence : Shareware, 39.95 $
Plateformes : Windows
Site : `http ://www.skyjuicesoftware.com/software/ds_info.html`

F5 Andreas Westfeld
Version : Release 11+
Licence : Open Source
Plateformes : Machine Virtuelle Java
Site : `http ://wwwrn.inf.tu-dresden.de/~westfeld/f5.html`

Hermetic Stego Hermetic Systems
Version : 7.51
Licence : Shareware, 14.99 €
Plateformes : Windows
Site : `http ://www.hermetic.ch/solo/hst.htm`

Hiderman Magic Voltige
Version : 3.0
Licence : Shareware, 35 $
Plateformes : Windows
Site : Plus de site officiel

InPlainView Justin Weiler
Version : 1.0
Licence : Shareware
Plateformes : Windows
Site : Plus de site officiel

InThePicture Intar technologies
Version : 2.2
Licence : Shareware, 25 $
Plateformes : Windows
Site : Plus de site officiel

Invisible Secrets NeoByte Solutions
Version : 4.6
Licence : Commercial, 39.95 $
Plateformes : Windows
Site : `http ://www.invisiblesecrets.com/`

Invisible Secrets NeoByte Solutions
Version : 2.1
Licence : Freeware
Plateformes : Windows
Site : `http ://www.invisiblesecrets.com/ver2/`

iSteg hanynet.com
Version : 1.5
Licence : Open Source
Plateformes : Mac
Site : `http ://www.hanynet.com/isteg/`

Jsteg Derek Upham
Version : -
Licence : Open Source
Plateformes : Windows
Site : plus de site officiel

Outguess Niels Provos
Version : 0.2
Licence : Open Source
Plateformes : Linux
Site : `http ://www.outguess.org/`

Pict Encrypt Pariahware
Version : 2.0
Licence : Freeware
Plateformes : Mac
Site : `http ://www.pariahware.com/pictencrypt.php`

Privacy Suite 2008 Steganos
Version : 10.0
Licence : Commercial, 59.95 €
Plateformes : Windows
Site : `http ://www.steganos.com/fr/produits/data-security/privacy-suite/`

Qtech H-V KIT STEGROUP
Version : 1.0
Licence : -
Plateformes : Windows
Site : `http ://www.datahide.com/BPCSe/QtechHV-program-e.html`

Snow Darkside Technologies
Version : 1.1
Licence : Open Source
Plateformes : Machine virtuelle Java
Site : `http ://www.darkside.com.au/snow/`

StegHide Stefan Hetzl
Version : 0.5.1
Licence : Open Source
Plateformes : Linux
Site : `http ://steghide.sourceforge.net/`

WbStego4Open -
Version : 4.0
Licence : Open Source
Plateformes : Windows et Linux
Site : `http ://wbstego.wbailer.com/`

C Abréviations

ASCII	American Standard Code for Information Interchange. Norme de codage de caractère largement répandue.
BMP	Contraction de Bitmap. Image dans laquelle les pixels sont stockés sous forme d'une matrice de point.
BPCS	Bit Plane Complexity Segmentation. Technique stéganographique se basant sur la complexité d'un groupe de pixels pour la dissimulation d'informations.
DCT	Discrete Cosine Transform. Opération mathématique permettant de transformer des points dans le domaine spatial, en information de fréquences équivalentes.
HTML	Hypertext Markup Language. Langage à balise utilisé pour la représentation de contenu sur Internet.
JFIF	JPEG File Interchange Format. Format de fichier le plus utilisé pour le stockage d'image utilisant la norme de compression JPEG.
JPEG	Joint Photographic Expert Group. Norme de compression pour les images numériques.
LSB	Least Significant Byte. Technique stéganographique utilisant les bits de poids faible d'une image pour dissimuler des informations.
MD5	Message Digest 5. Fonction de hashage permettant d'obtenir une empreinte numérique d'un fichier.
PDF	Portable Document Format. Langage de description de document crée par Adobe Systems.
PNG	Portable Network Graphic. Format ouvert d'images numériques défini pour remplacer le format GIF.
WAV	WAVEform audio format. Standard pour le stockage des données audio numériques.

D Pré-projet de diplôme

Première partie

Introduction

1 Introduction à 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 leur propre domaine de compétence. Dans certaines situations, le faite même de vouloir transmettre des données de manière chiffrée sera jugé comme suspect. De la même manière, de plus en plus de pays mettent en place de forte restriction concernant la longueur des clés cryptographiques, ainsi que la cryptographie en elle même.

Le regain d'intérêt actuel pour la stéganographie provient de ces restrictions imposée 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.

Personne ne sachant réellement répondre à la question "Quelles informations sont-elles réellement récoltée sur moi sur Internet ?", la stéganographie vient comme un moyen de pouvoir avoir contrôle de ce que nous laissons transparaître vers l'extérieur. A une époque ou l'on arrive plus à estimer la puissance de certaine agence de renseignement, ni à connaître leur véritable capacité, il semble légitime de s'assurer que notre sphère privée soit respectée.

Pour illustrer le domaine d'application de la stéganographie, cette dernière est souvent introduite par le problème des prisonniers [2].

Deux prisonniers souhaitent établir un plan d'évasion. Pour ce faire, ils ont la possibilité de se transmettre des messages. Cependant, ces messages transitent à travers le gardien, qui a donc accès au contenu. Tout contenu jugé inapproprié sera détruit, et pourra engendrer une lourde sanction. Dans cette optique, le contenu doit donc paraître anodin au yeux du gardien. Cette situation rend inutilisable la cryptographie, car un contenu indéchiffrable attirera l'attention du géolier.

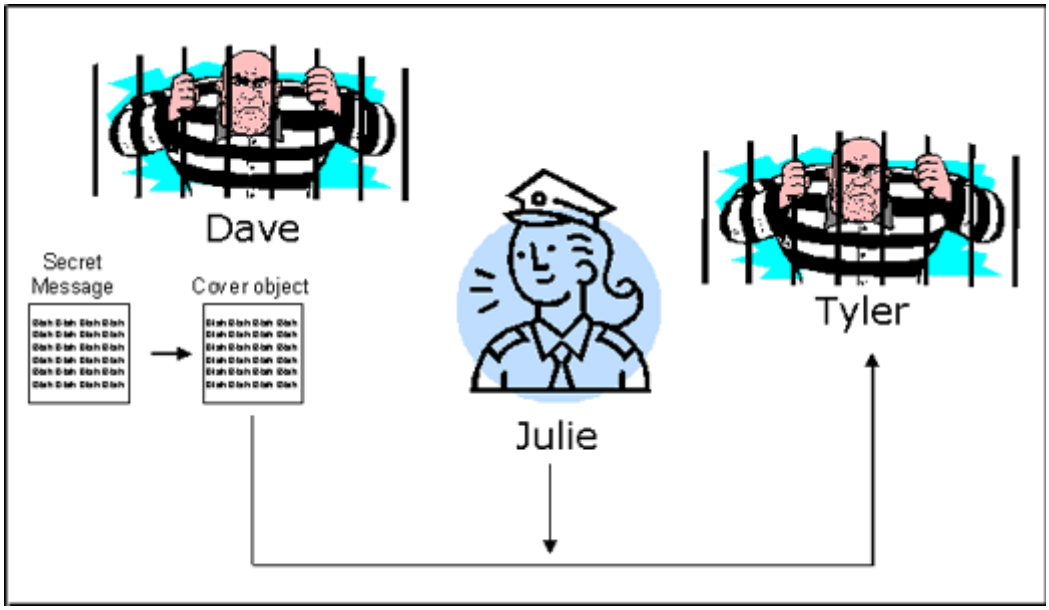

FIG. 30 – Problème des prisonniers [2]

La stéganographie a donc comme objectif de pouvoir entretenir des communications sécurisée, sans pour autant attirer l'attention d'autrui.

La stéganographie connaît ses premiers prémices à l'antiquité. A cette époque, les messages étaient écrit sur le crâne rasé d'un esclave. Une fois que ses cheveux avait repoussés, il pouvait transmettre le message sans se faire inquiéter.

D'autres méthode bien connues sont apparues par la suite. Ainsi, l'encre invisible eu grand succès il y a quelques décennies. Le principe était d'écrire le message devant rester secret avec une encre invisible (jus de citron, urine, etc...), puis d'écrire cette fois-ci avec une encre bien visible un message au contenu anodin. Une fois arriver à destination, un traitement particulier permettait de recouvrir le message.

De nos jours, la stéganographie à pris de l'ampleur dans des supports bien particulier, les supports numériques. Que ce soit des fichiers audio, vidéo ou image, il représente des supports privilégié pour la transmission d'information. Avec la généralisation d'Internet, le volume de données qu'il représente est énorme. Cela représente autant de support possible à la stéganographie.

Les attentats du 11 Septembre 2001 ont relancé l'intérêt des milieux académiques et professionnels pour la stéganographie. L'utilisation soupçonnée de techniques stéganographiques pour l'organisation de ces méfaits par la nébuleuse Al-Qaïda, en serait le principal facteur[7].

Ce document est organisé en trois parties. La première partie parle de généralité concernant la stéganographie. La deuxième partie détaille quelques techniques stéganographique largement utilisé sur les supports numériques. En troisième et dernière partie, quelques techniques de stéganalyse seront exposées.

[7] Aucune preuve à ce jour n'a pu être trouvée confirmant cette hypothèse.

2 Architectures

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 suivant :

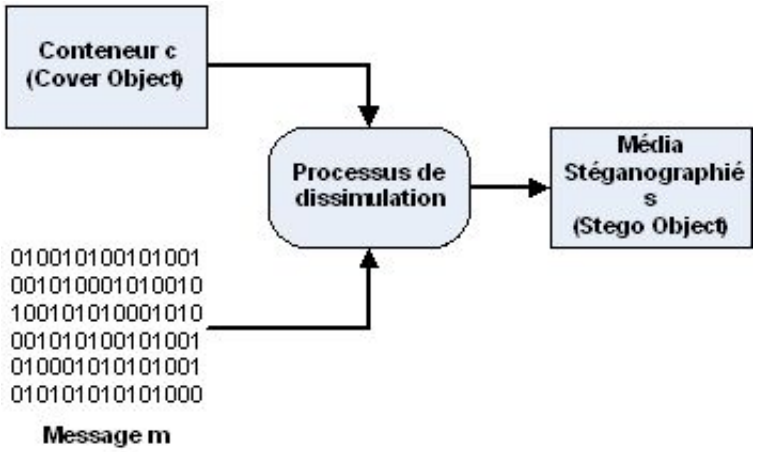

Fig. 31 – Schéma simplifié de fonctionnement

Il existe trois type de protocoles de stéganographie, correspondant de près à ce qui existe en cryptographie.

La stéganographie pure est un système dans lequel le secret de dissimulation des données ne réside que dans l'algorithme utilisé à cet effet. La découverte de cet algorithme rompt la dissimulation de la communication. Ceci revient à mettre en place de la "sécurité par l'obscurité".

La stéganographie à clé secrète est similaire à la cryptographie symétrique, l'échange de données confidentielles nécessite, au préalable, l'échange d'une clé secrète que l'on ne partagera que avec notre interlocuteur. Il est donc nécessaire d'avoir un canal sécurisé, ou de rencontrer en personne notre interlocuteur, afin d'être certain que cette dernière ne soit pas compromise. Cette clé aura une influence sur la manière de "cacher" l'information.

La stéganographie à clé public, quant à elle, est similaire à la cryptographie asymétrique. La personne voulant envoyer des données à un autre interlocuteur, sans éveiller de soupçons, utilisera la clé public de ce dernier. La clé public étant à priori connue de tout le monde, il n'y aura pas besoin d'échange préalable "sécurisé". La personne recevant ce message sera la seul à pouvoir en extraire son contenu à l'aide de sa clé privée.

Voici un schéma plus complets du processus stéganographique :

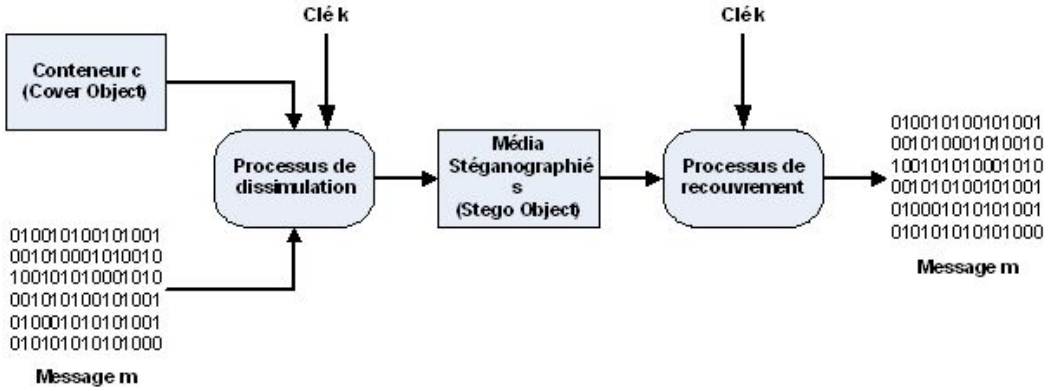

FIG. 32 – Schéma complet

3 Caractéristiques

Trois critères permettent de classer les algorithmes stéganographique : La Capacité, la transparence et la robustesse.

La Capacité correspond à la masse de données qui peut être insérée dans un conteneur, relativement à la taille de celui-ci.

La transparence permet de quantifier le bruit généré par le processus de dissimulation, et par la même l'invisibilité de notre message.

Pour finir, la robustesse spécifie la capacité qu'à notre message stéganographié de rester intacte après que le conteneur aie subit des modifications (filtrage, etc...).

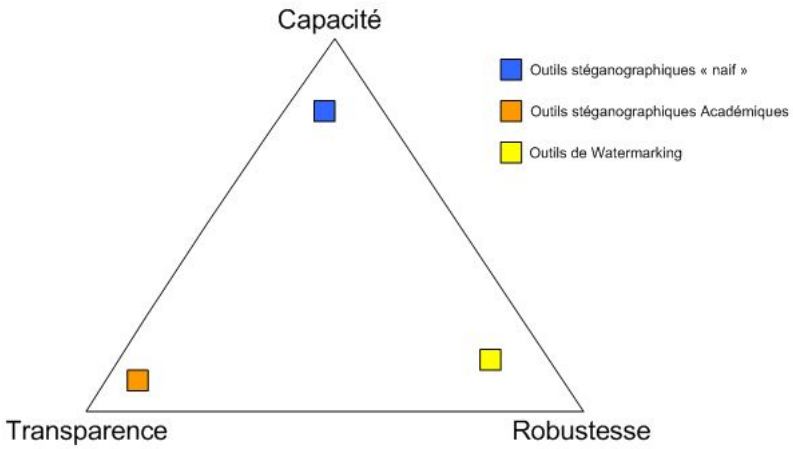

FIG. 33 – Triangle des caractéristiques

Ces trois critères ne peuvent pas être maximisé simultanément. Chacun d'entre eux aura une influence sur l'autre. Par exemple, la capacité va en contradiction avec la transparence. Sur la figure 33, des outils ont été placés afin d'en définir la caractéristique principale.

Les outils de stéganographie dit naïfs correspondent à la grande majorité des outils disponibles sur internet. Ils cachent les informations dans les conteneurs sans réellement se préoccuper de la facilité à détecter ces données, ni les influences que ces données peuvent avoir sur le conteneur d'un point de vue statistique.

Les outils de stéganographie académique sont quant à eux développé par des équipes de recherches (notamment l'équipe de Fridrich). Leur objectif est de faire évoluer en parallèle stéganographie et stéganalyse. Leur objectif principal est d'arriver à des algorithmes totalement transparents (pour les méthodes actuelles), afin de pouvoir en déduire des méthodes de stéganalyse encore plus performantes. De récentes recherche portent sur la maximisation de l'espace de dissimulation disponible. Ces outils arriveront peut-être à allier transparence à capacité dans un avenir proche.

Pour finir, les outils de watermarking, utilisé principalement pour la protection de droit d'auteur, sont principalement développé afin d'avoir une très grande robustesse. Contrairement à la stéganographie, leur adversaires sont de type actif. Le contenu du watermarking ne leur servant en rien, leur unique objectif est la suppression pur et simple de ce dernier.

Deuxième partie

Techniques Stéganographiques

Cette partie va s'atteler à détailler quelques techniques stéganographique utilisée couramment dans les images. A noter que certaines de ces techniques sont transposable aux autres domaines que sont l'audio et la vidéo.

Selon [16], il est possible d'établir une hiérarchie au niveau des techniques stéganographique. En partant du moins sécurisé, cette hiérarchie serait :

1. L'ajout du message à la fin du fichier.
2. L'ajout du message dans les espaces inutilisés du conteneur.
3. Ajout du message dans les données de l'image de manière séquentielle.
4. Ajout du message dans les données de l'image en utilisant une séquence pseudo-aléatoire.
5. Ajout du message dans les données de l'image en utilisant une séquence pseudo-aléatoire, en prenant soin de modifier les données inutilisées afin de ne pas être visible d'un point de vue statistique.

Ces cinq points peuvent être regroupé en deux catégories distinctes. Pour les deux premiers points, il correspondent à la technique dite de fusion. Pour ce qui est du reste, il peuvent être regroupé dans ce qui a trait à la modification LSB.

Ces deux techniques vont être détaillée ci-dessous.

4 LSB

La technique du LSB, signifiant Least Significant Bit, est de loin la technique la plus répandue. Son succès provient d'une grande facilité de mise en oeuvre, ce qui permet d'en trouver de nombreuses implémentations.

Sous l'appellation LSB est regroupé tout ce qui a trait à la dissimulation de données par la modification du bit de poids faible d'un élément. Cela va de la valeur d'un pixel, jusqu'à la modification de la valeur d'un coefficient DCT dans le cas de la norme JPEG. Tous se base sur l'insensibilité du système visuel humain a un faible changement de couleurs.

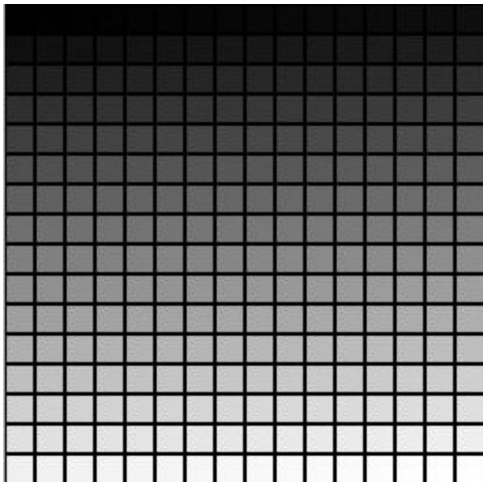

FIG. 34 – mire de 256 niveaux de gris

Comme cela peut être mis en évidence à l'aide de la figure 34, le changement du bit de poids faible correspond à un déplacement horizontal d'une case dans la mire. Aucun changement n'est perceptible.

Plusieurs types de modifications peuvent être effectuées sur ces LSB. La plus répandue consiste simplement à remplacer ces bits par les bits du message que l'on souhaite dissimuler. Appelée "LSB Replacement" dans la littérature, cette technique semble à première vue très efficace, cependant comme cela sera détaille plus loin dans le document, elle possède le gros désavantage de modifier de manière significative les statistiques du conteneur.

Un exemple de cette méthode peut être donner à l'aide de la matrice[8] C suivante représentant une conteneur de 4 élément sur 4.

$$C = \begin{bmatrix} 00 & 00 & 10 & 10 \\ 01 & 11 & 10 & 00 \\ 00 & 11 & 10 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix}$$

Pour simplifier, chaque élément est codé sur 2 bits uniquement. Si l'on souhaite dissimuler le message m dans notre conteneur, une première remarque sera sur la taille maximum du message qui sera égal au nombre d'éléments dans la matrice. Cela est vrai pour autant que l'algorithme se limite à la modification du seul bit de poids faible.

$$m_{max} = 1001\,1010\,0011\,1001$$

Dans le cas du conteneur C, la taille maximal est donc de 16 bits. La matrice stéganographié résultant du processus sera

$$S_{replacement} = \begin{bmatrix} 01 & 00 & 10 & 11 \\ 01 & 10 & 11 & 00 \\ 00 & 10 & 11 & 01 \\ 01 & 00 & 00 & 01 \end{bmatrix}$$

[8] Matrice de pixels, ou matrice de coefficients DCT

En comparant S à C, on remarque que l'opération consiste donc uniquement en une surécriture du bit de poids faible.

La seconde méthode est appelée "LSB Matching". Elle diffère de ce qui précède par le fait qu'elle ne modifie pas obligatoirement tous les bits de poids faible.

Le principe consiste à comparer la valeur du bit de poids faible à la valeur du bit à dissimuler. Si ils correspondent, aucun changement n'est effectué. Dans le cas contraire, une incrémentation/décrémentation de manière aléatoire de la valeur de l'élément de 1 sera effectuée. Cela aura pour incidence de codé la valeur désirée au niveau du LSB.

En reprenant la même matrice C que précédemment,

$$C = \begin{bmatrix} 00 & 00 & 10 & 10 \\ 01 & 11 & 10 & 00 \\ 00 & 11 & 10 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix}$$

dans laquelle le message m_{max} suivant est à dissimuler.

$$m_{max} = 1001\,1010\,0011\,1001$$

Le résultat de l'application de cette méthode à la matrice de base C sera cette fois

$$S_{matching} = \begin{bmatrix} 01 & 00 & 10 & 01 \\ 01 & \mathbf{00} & 11 & 01 \\ 00 & 10 & 11 & \mathbf{11} \\ 01 & 00 & 00 & \mathbf{11} \end{bmatrix}$$

Cette matrice est un des résultats possible, le choix entre l'incrémentation et la décrémentation étant fait de manière aléatoire. En gras sont mis en évidence des erreurs pouvant provenir du processus. Par exemple, l'incrémentation d'une valeur 11 aura pour conséquence sont passage à 00. L'écart des valeurs étant trop élevé, le résultat de l'opération risque d'être visuellement décelable. L'algorithme doit donc veiller à ce que ce genre de situation ne soit pas autorisée (autorisé uniquement l'incrémentation lorsque la valeur est la plus petite possible par exemple).

Comparativement à la méthode précédente, cette dernière est moins sensible aux analyses statistiques. Du faite de l'incrémentation/décrémentation aléatoire des valeurs des éléments, cette méthode n'ajoutera pas les mêmes distorsions statistiques que le "LSB Replacement".

Dans les deux cas, le choix de l'image hôte est un point essentiel dans l'optique d'obtenir la meilleure transparence possible. Les images contenant peu de couleurs sont a proscrire. Certains experts recommandent l'utilisation d'image en niveau de gris comme meilleurs conteneurs. Il conseille l'utilisation d'images non compressées.

Par la suite vont être détaillées les deux plus grands domaines d'application de la technique du LSB que sont, d'une part, le domaine spatial et de l'autre, le domaine de la transformée en cosinus discret.

4.1 Domaine Spatial

Dans le domaine spatial, le dissimulation du message est directement effectué au niveau du codage des pixels. Une image peut être représentée à l'aide d'une matrice de pixel. Chaque pixel est représenté à l'aide de 1 à 32 bits. Ce nombre dépend de la représentation des couleurs utilisées. Les plus utilisées sont cependant :

– Sur 8 bits
– Sur 24 bits

Sur 8 bits, deux méthodes de codage sont utilisées. Dans le cadre d'une image en niveaux de gris, chaque pixel code directement le niveau de gris approprié. Dans celui d'une image couleurs, on utilise le mécanisme d'image indexée. Au sein de chaque pixel est codé une valeur correspondant à l'index de la couleur à utilisé dans la palette de couleurs définie.

Pour ce qui est du codage sur 24 bits, on utilise généralement le RGB (Red Green Blue) pour la représentation des couleurs. Chaque pixel est codé à l'aide d'un triplet de byte spécifiant chacune des composantes principales. Dans ce type de représentation, il est possible de dissimuler 3 bits par pixel (1 par composante) sans aucun impact visuel.

FIG. 35 – A gauche, image originale. A droite, image contenant un PDF de 32 Kb dissimulé à l'aide d'Invisible Secrets 4

Les fichiers généralement utilisé sont au format BMP et GIF.

Cette technique est très utilisée car elle facile à mettre en oeuvre. Elle permet de directement ajouter les données à l'image, sans avoir à passer par un mécanisme de compression/décompression comme ce serait le cas pour la modification des coefficients DCT.

Crée sa propre implémentation devient très facile, cela permet de s'affranchir de l'utilisation d'un produit existant et par la même occasion, d'éviter la détection de signature étant attribuée à ce programme (technique encore très utilisée dans les outils de détection). De plus, il est possible de vraiment contrôler la manière dont seront dissimulé les données au sein du conteneur. Il sera possible de facilement implémenté des mécanismes permettant de déjouer les tentatives d'analyse statistiques.

L'inconvénient majeur provient des formats utilisés. Les images au format JPEG sont, de loin, les plus répandue sur Internet. Le simple fait d'envoyé une image au format BMP

peut attirer l'attention, et de ce fait, mettre à néant tous les efforts de dissimulation de données.

Ainsi, les outils permettant la dissimulation de données dans le format JPEG constitue une des meilleurs solutions offerte. La prochaine section va en détailler l'idée générale.

4.2 Domaine de la Transformée en Cosinus Discret (DCT)

Comme souligné dans la section précédente, les images au format JPEG [8] représentent la grande majorité des images circulant sur le réseau des réseaux. Une image envoyé à son collègue en utilisant ce format est devenu quelque chose d'anodin. Dans ce contexte, ce format semble être un conteneur de choix pour des communications secrètes. La dissimulation d'information dans des formats de compression à perte se révèle, cependant, plus difficile. Ce type de compression utilisant les mêmes données redondantes (n'ayant aucun impact sur la perception du média) que celle utilisée dans le processus de dissimulation.

Afin d'assurer un maximum de transparence, les modifications doivent avoir lieu dans le domaine des Transformées en Cosinus Discret [18] et non dans le domaine spatial. D'un point de vue compression cela permet d'éliminer les hautes fréquences (saut brusque de couleur) qui ne sont que difficilement décelable par l'oeil humain.

En ce qui concerne la stéganographie par modification des coefficients DCT, elle intervient après la phase de quantification de l'algorithme de compression (voir figure 36).

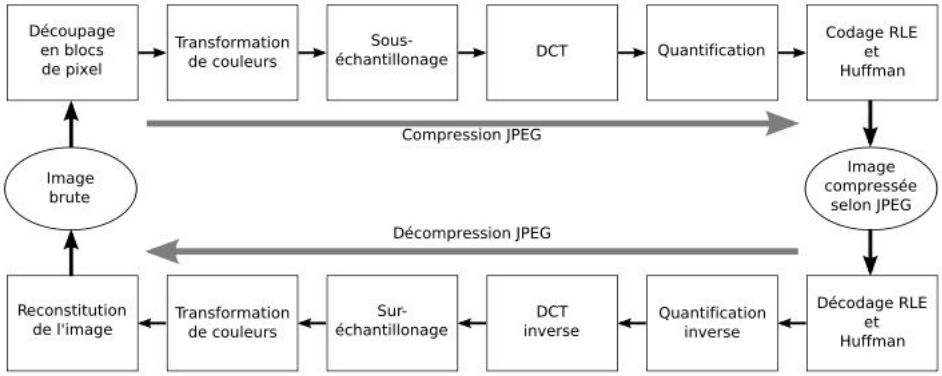

Fig. 36 – Schéma Bloc du processus de compression/décompression JPEG[8]

Pendant ce processus, la plupart des coefficients étant ramenés à zéro, tout effort de dissimulation intervenant avant la quantification serait rendu inutilisable. Un autre facteur à prendre en compte, et de ne pas modifier les propriétés de compression. Après quantification, les matrices de coefficients passe par un algorithme de compression sans perte, le codage de Huffmann. Afin de ne pas modifier les informations, aucun coefficient étant égal à 1 ou 0 ne devra être modifier. Si l'on remplace un coefficient à 1 par 0, cela engendrera un meilleurs taux de compression, ce qui n'est pas souhaité.

De cela découle un des gros inconvénient de cette méthodologie. Une matrice de coefficients étant principalement composée de 0, et ceux-ci ne pouvant pas être modifiés, la capacité du

conteneur s'en retrouve fortement réduite. De plus, contrairement au modification dans le domaine spatial, implémenté une telle technique se révèle plus ardu. Il est en effet nécessaire de coder tout le compresseur JPEG, ce qui n'est pas une tâche aisée. La dissimulation dans les coefficients DCT ne se révèle pas non plus plus performante d'un point de vue transparence. Quelques méthodes de stéganalyse dans le domaine spatial on rapidement été adaptées au domaine DCT [1].

Malgré ces inconvénients, la très forte distribution de ce format est un avantage majeur. De nombreuse personne (surtout du domaine académique) l'on compris, est des outils, tel F5 ou Outguess ont vu le jour. Ce dernier va être détaillé dans la section suivante.

4.2.1 Algorithme Outguess

Développer par Niels Provos, à qui l'on doit notamment le document [1], cette algorithme avait comme objectif de passer totalement inaperçu lors d'une analyse statistique des χ^2. Il travaille sur les fichiers de type JPEG en effectuant une surécriture des LSB au niveau des coefficients DCT [8]. Cependant, pour ne pas modifier les propriété de compression, il modifie uniquement les coefficients étant différent de 1 ou 0.

Afin de paraître invisible lors d'analyse statistique du première ordre (analyse des histogrammes de valeur DCT), l'algorithme opère en deux étapes

En première passe, il modifie les LSB des coefficients DCT de manière pseudo-aléatoire. Ceci est défini à l'aide d'une clé.

En deuxième passe, il parcours les coefficients DCT non modifiés afin d'adapter leur valeur de tel sorte que l'histogramme des coefficients après modification soit égal à celui du fichier d'origine.

Afin de pouvoir être sûr de retrouver l'histogramme d'origine après la manipulation, il effectue un calcul de la taille maximal des données à dissimuler en fonction de l'image. Ceci est, bien sur, effectué avant le début des opérations.

FIG. 37 – A gauche, image originale. A droite, image contenant le fichier outguess.h stéganographié avec Outguess

Comme il peut être remarquer sur la figure 37, aucun différence n'est visuellement apparente. Fridrich [9] a défini une méthode permettant de détecter de manière viable les contenus stéganographié à l'aide de Outguess.

5 Fusion

Cette technique, que l'on peut considérer comme de la stéganographie naïve, consiste à ajouter les données à cacher au fichier. Pour ce faire, cette méthode utilise des emplacements inutilisés ou non lu par la plupart des décodeurs d'image.

On distingue deux fonctionnements : L'ajout de données en fin de fichier et l'ajout au niveau des en-têtes de fichier.

L'ajout en fin de fichier est rendu possible par le fait que la plupart des décodeurs d'image ne lisent pas le fichier image dans son ensemble. Pour la plupart des formats d'image disponible, un certaine chaîne de bits est définie afin de marqué la fin de l'image. L'ajout en fin d'image append simplement après cette chaîne les données dissimulées. Aucune limitation de taille n'est imposée, cependant un fichier image de 20 Mbytes risque de ne pas passer inaperçu.

Pour ce qui est de l'ajout dans les en-tête, certains formats comme le bitmap définisse un champ permettant de spécifier l'offset à partir du quel l'image commencera. En spécifiant un offset un peu plus long il est possible de cacher entre deux les données à dissimuler. Cela peut aussi être fait à l'aide d'ajout de commentaire, pour le JPEG par exemple.

FIG. 38 – A gauche, image originale. A droite, image contenant des données dissimulées dans le champs de commentaire à l'aide d'Invisible Secrets 4

La figure 38 montre l'ajout de données en temps que commentaire dans une image JPEG. Pour confirmer qu'aucune limitation de taille n'est imposée, un fichier PDF de 1.57 MByte à été dissimulé sans problème.

Malgré le faite qu'aucune limitation de taille n'est imposée pour cette technique, elle tiens plus du gadget que de la stéganographie. Une simple vérification de la consistance des fichiers, avec des vérifications ciblées au endroits permettant la dissimulation permet d'éradiquer toutes tentatives de communication cachée.

Troisième partie

Stéganalyse

Contrairement à la cryptanalyse, dont le but est de récupérer les données ayant été cryptée, la stéganalyse n'a pas comme objectif de retrouver les données dissimulées à l'aide d'un algorithme stéganographique. Elle consiste uniquement en la détection de contenu stéganographié, ce qui n'est déjà pas une mince affaire. En détectant si un média sert de conteneur stéganographique, des artefacts laissé par un algorithme particulier pourront éventuellement être retrouvé. En connaissance de cet algorithme, il sera possible de lancer une attaque par dictionnaire, voir force brute, afin de déterminer la clé stéganographique.

Plusieurs catégories de méthodes peuvent être utilisée dans cette optique. La plus simple consiste en la détection de signature d'un logiciel donné. Certains programmes laissent derrière eux, de manière intentionnelle ou par erreur de conception, des artefacts permettant de caractériser leur passage. Cette façon de faire possède l'avantage de directement fournir le nom de l'application utilisée pour la dissimulation de donnée. Par contre, chaque application doit faire l'objet d'une analyse spécifique.

La détection d'irrégularité statistique est une autre catégorie d'analyse. Elles se basent sur la quantification de distorsion du média analysé, comparativement à des distributions statistiques théorique représentant un média de base. Sa grande force est de pouvoir détecter un large panel d'application se basant sur la même technique stéganographique.

La catégorie permettant le scope de détection le plus large est donnée par les outils de stéganalyses universelles. Se basant sur des réseaux neuronaux, leur capacité dépend uniquement de la qualité de la base d'entraînement, ainsi que du bon choix des caractéristiques sensées qualifiée le média.

Dans ce document, seul des analyses statistiques seront détaillées. Pour ce qui est de la stéganalyse universelle, une courte description sera données en seconde partie.

6 Techniques Spécifiques

6.1 Analyse du χ^2 [3]

Dans leur document [3], Andreas Westfeld et Andreas Pfitzmann mettent en évidence que la surécriture des LSB dans les fichiers images en modifie les caractéristiques d'un point de vue statistique.

En effet, cela à pour conséquence de créer ce que les auteurs ont appelé *pairs of values* (PoV). En partant du principe que les bits du message à transmettre soient uniformément distribué, ces PoVs auront des fréquences d'apparition quasi identique, contrairement à une image (voir Figure 39).

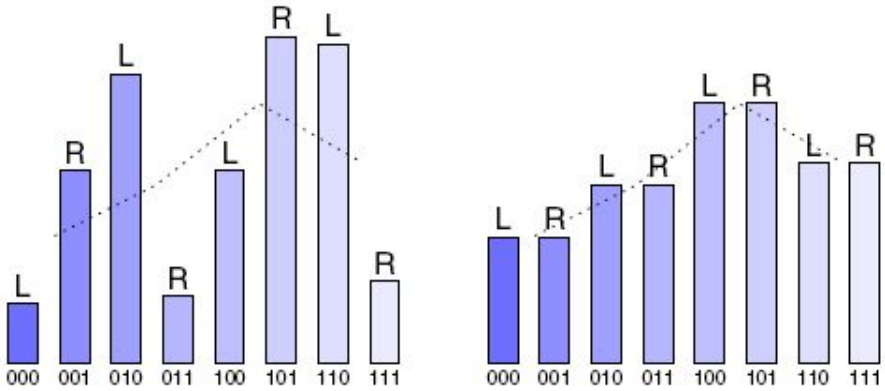

FIG. 39 – A gauche, image original et à droite, image avec surécriture des LSB [3]

Sur la figure précédente, la ligne en traitillé correspond à la moyenne des deux valeurs composant un PoV. Comme on peut le remarquer, cette moyenne n'est pas affectée par la manipulation. Cette moyenne sera donc utilisée afin de crée un modèle théorique à partir d'un fichier dont on ne sait pas si il a été modifié. La moyenne théorique sera donc calculée à partir de la formule suivante

$$n_i^* = \frac{n_{2i} + n_{2i+1}}{2} \tag{1}$$

Pour chaque paire de valeurs i. n correspondant à la fréquence d'apparition de la valeurs dans l'image.

La différence de distribution entre les valeurs calculées théoriquement est celle étant réellement présentes dans le fichier image permettra de définir la probabilité qu'un contenu caché soit présent dans le fichier.

La similarité entre les deux distributions peut être déterminé en utilisant le test du χ^2 qui, dans ce cas, s'exprime ainsi

$$\chi^2_{k-1} = \sum_{i=1}^{k} \frac{(n_i - n_i^*)^2}{n_i^*} \tag{2}$$

ou $k-1$ représente le degré de liberté. Cela correspond au nombre de paires (PoV) qui ont été définies précédemment.

La probabilité que les deux distributions soient identiques est donné par l'expression suivante

$$p = 1 - \frac{1}{2^{\frac{k-1}{2}}\Gamma(\frac{k-1}{2})} \int_0^{\chi^2_{k-1}} e^{-\frac{x}{2}} x^{\frac{k-1}{2}-1} dx \tag{3}$$

avec Γ représentant la fonction gamma d'Euler qui s'écrit

$$\Gamma(z) = \int_0^{+\infty} t^{z-1} e^{-t} dt \tag{4}$$

La probabilité résultante correspondra à la probabilité qu'il y aie un contenu caché dans l'image. Le calcul de la probabilité s'effectue en plusieurs étapes. Le calcul commence par un échantillon représentant une petite partie de l'image (en commençant par le début). A chaque étape, on ajoute de manière séquentielle une partie de l'image à notre échantillon, jusqu'à arriver au bout de l'image. Cela évite d'avoir une probabilité faible lorsque la taille des données cachées est faible. Cela permet aussi d'avoir une estimation de la taille des données dissimulées (la probabilité ne chutant pas directement à 0, le résultat sera biaisé. voir figure 40).

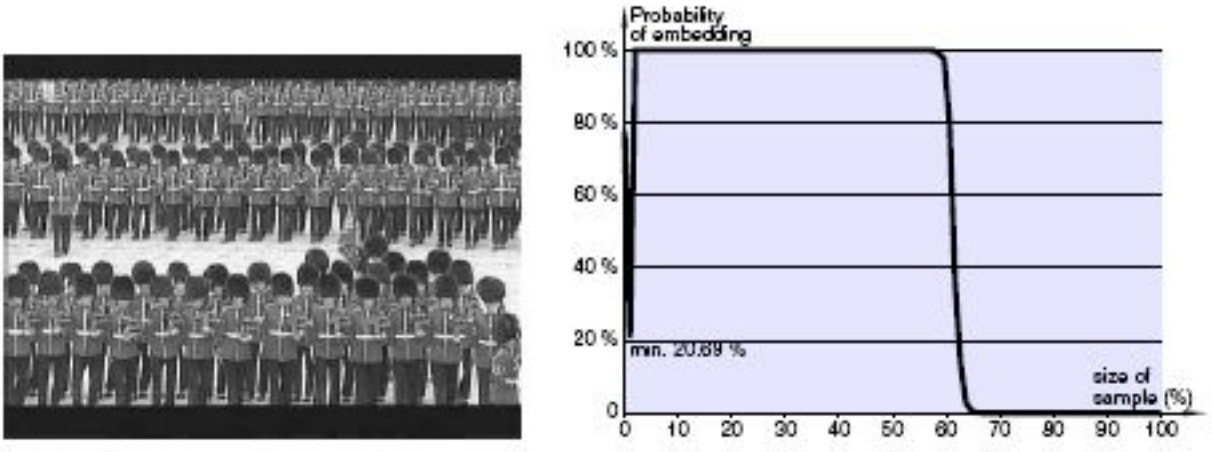

FIG. 40 – Résultat avec 50% de la capacité [3]

Si l'on applique cette méthode à l'exemple LSB de la partie 2, il serait possible de mieux visualiser l'application de cette méthode à une image. Pour rappel, voici l'image 4x4 exemple :

$$C = \begin{bmatrix} 00 & 00 & 10 & 10 \\ 01 & 11 & 10 & 00 \\ 00 & 11 & 10 & 00 \\ 00 & 00 & 00 & 00 \end{bmatrix} max = 1001\,1010\,0011\,1001 \;\; S_{replacement} = \begin{bmatrix} 01 & 00 & 10 & 11 \\ 01 & 10 & 11 & 00 \\ 00 & 10 & 11 & 01 \\ 01 & 00 & 00 & 01 \end{bmatrix}$$

Les *pairs of values* seront [00,01] et [10,11]. La première étape consiste à observer le nombre d'occurrences de chaque élément. Chaque élément sera classé à l'aide d'un indice suivant le tableau ci-dessous.

indice	élément
0	00
1	01
2	10
3	11

TAB. 19 – Correspondance entre indice et élément

En classant les éléments de nos deux images (image originale et image stéganographiée) dans les divers indices, cela donne le tableau d'occurrences suivant.

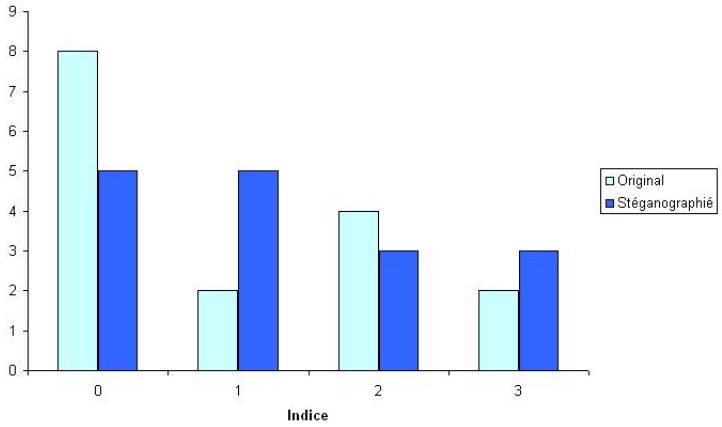

FIG. 41 – Graphique du nombre d'occurrences

indice	0	1	2	3
Originale	8	2	4	2
Stéganographiée	5	5	3	3

TAB. 20 – Occurrences de chaque élément

Ensuite, à l'aide de la formule (1), la distribution des occurrences théorique peut être calculée. Dans cette exemple formée de 2 paires de valeurs, elle sera formée de deux valeurs.

$$n_0^* = \frac{n_0 + n_1}{2} = 5$$

$$n_1^* = \frac{n_2 + n_3}{2} = 3$$

Ensuite, avec la formule (2), il est possible d'appliquer le test du χ^2à notre image stéganographiée, ainsi qu'à notre image originale. Le calcul s'effectuant sur les pairs de valeurs, pour l'exemple, seul les valeurs impaires d'indices ont été prise en compte (pour la première paire de valeurs, seul l'indice 1 à été utilisé pour le calcul).

$$\chi^2_{k-1_{Stegano}} = \sum_{i=1}^{k} \frac{(n_i - n_i^*)^2}{n_i^*} = \frac{(5-5)^2}{5} + \frac{(3-3)^2}{3} = 0$$

$$\chi^2_{k-1_{Originale}} = \sum_{i=1}^{k} \frac{(n_i - n_i^*)^2}{n_i^*} = \frac{(2-5)^2}{5} + \frac{(2-3)^2}{3} = \frac{32}{15}$$

Il est ensuite possible de calculer la probabilité d'avoir un contenu dissimulé via l'équation (3). Le résultat du χ^2 intervenant dans les bornes d'intégration de la fonction de densité, en ce qui concerne l'image stéganographiée, le résultat est aisément identifiables. Le résultat de l'intégration étant nul, le membre de droite de la soustraction s'annule. Cela donne donc une probabilité de 1 d'avoir affaire à un contenu stéganographié.

$$p_{stego} = 1 - \frac{1}{2^{\frac{k-1}{2}}\Gamma(\frac{k-1}{2})} \int_0^0 e^{-\frac{x}{2}} x^{\frac{k-1}{2}-1} dx = 1 - 0 = 1$$

En ce qui concerne le même calcul pour l'image originale, le même raccourci n'est pas possible. Il faut donc calculer l'entier de l'équation.

$$p_{originale} = 1 - \frac{1}{2^{\frac{1}{2}} * 1.772} \int_0^{\frac{32}{15}} e^{-\frac{x}{2}} x^{-\frac{1}{2}} dx = 0.144$$

Il y a donc une probabilité de 14.4 % que notre image originale dissimule des données, ce qui est bien sûr erroné. Ce biais provient certainement de l'exemple de taille trop faible. Une image 4x4 est trop petite pour permettre d'avoir un nombre d'occurrences assez grand. Dans la méthode originale, il est spécifié de regrouper les paires de valeurs n'ayant pas un nombre d'occurrences assez élevé.

Il est a noter que dans cette exemple le calcul a été effectué sur l'ensemble de l'échantillon. Vu la faible taille de l'image d'exemple, l'application de la méthode à des échantillons plus réduit semble illusoire. Cependant, sur des images de taille normales, cette manière de faire permet des résultats bien meilleures.

En conclusion, cette méthode donne de bon résultat, mais a un champ d'application très faible. Elle permet uniquement la détection de données cachées de manières séquentielle au niveau des LSB. Ainsi, l'utilisation d'une séquence pseudo-aléatoire afin de déterminer

les valeurs modifiées rends caduque l'utilisation de cette méthode. De plus, elle ne permet qu'une estimation de la taille des données approximative.

Dans [1], cette méthode à été adaptée avec succès au image JPEG. Ceci à été rendu possible en appliquant la même logique que ci-dessus, mais cette fois-ci sur les coefficients DCT [8] des images JPEG.

Niels Provos [1] adapte la méthode précédente afin de pouvoir détecter le contenu cachées de manière aléatoire. Pour ce faire, il n'utilise pas une taille d'échantillon s'accroissant sans cesse depuis le début du fichier, mais déplace la fenêtre d'observation.

6.2 Analyse RS

Cette méthode, développée par Fridrich, Miroslav Goljan et Rui Du [17], est basée sur la classification de groupes de pixels en catégories distinctes.

En se basant sur une image de $L \times H$ pixels, dont chaque pixel a une valeur comprise dans l'ensemble P. Pour une image en 256 niveaux de gris (8 bits), $P = \{0, ..., 255\}$. La première opération a effectuer, est de diviser l'image en groupes disjoints de n pixels adjacents $G = (x_1, ..., x_n)$. Ce nombre n est défini par les auteurs à 4. Une fonction de discrimination f permettant d'évaluer la rugosité de chacun des groupes G sera définie. Elle attribuera une nombre réel à chacun des groupes, plus le groupe sera bruité, plus cette valeurs sera grande. Un exemple de cette fonction de discrimination peut être donné par :

$$f(x_1, ..., x_n) = \sum_{i=1}^{n-1} | x_{i+1} - x_i | \tag{5}$$

Cette fonction peut être déterminée par rapport au propriétés statistiques de l'image. Pour la suite, la fonction (5) sera utilisée comme fonction de discrimination.

Ensuite, il est défini des fonction réversible sur l'ensemble P. Ces fonctions consiste essentiellement en des permutations de valeurs. Ces fonctions sont au nombre de trois :

$$F_1 : 0 \leftrightarrow 1,\ 2 \leftrightarrow 3,\ ...,\ 254 \leftrightarrow 255$$

$$F_{-1} : -1 \leftrightarrow 0,\ 1 \leftrightarrow 2,\ ...,\ 255 \leftrightarrow 256$$

$$F_0 : identite$$

Afin d'appliquer ces fonctions de permutation sur les groupes G de pixels définis plus haut, Il faut définir une matrice de permutation M, ainsi que la matrice inverse $-M$. Ces matrices auront une taille de $1 \times n$, avec des valeurs comprises dans $\{-1, 0, 1\}$. Cela définira le fonction de permutation à appliquer à chacun des membre du groupes de tel sorte de donner le groupe permuté $F(G) = (F_{M(1)}(x_1), F_{M(2)}(x_2), ..., F_{M(n)}(x_n))$.

Ceci étant défini, il est possible de classifier chaque groupe de pixels dans une des trois catégories distinctes. Ces catégories sont définies de la manière suivante :

$$\begin{aligned} \text{Groupe Régulier :} \quad & G \in R \Leftrightarrow f(F(G)) > f(G) \\ \text{Groupe Singulier :} \quad & G \in S \Leftrightarrow f(F(G)) < f(G) \\ \text{Groupe inutilisable :} \quad & G \in U \Leftrightarrow f(F(G)) = f(G) \end{aligned}$$

Les fonctions de permutation des valeurs des pixels simulent l'ajout de bruits à l'image source. Dans le cas d'une image, l'ajout de bruit aura comme influence une augmentation de la fonction de discrimination. Au niveau de la classification des groupes, cela représentera une augmentation de ceux dans le groupe R.

En définissant R_M comme le pourcentage de groupes réguliers après l'application des transformations de la matrice M, S_M en étant le pendant pour les groupes singuliers. De la même manière, il faut définir R_{-M} et S_{-M} pour la matrice $-M$. Il est possible d'en déduire les relations suivantes :

$$R_M + S_M \leq 1 \text{ et } R_{-M} + S_{-M} \leq 1$$

En se basant sur l'équation (6), il est possible de faire l'hypothèse que l'application de la matrice M ou de sa matrice inverse, ne va pas changer de manière significative la distribution des groupes. Donc sur une image, les relations $R_M \cong R_{-M}$et $S_M \cong S_{-M}$ devrait être vérifiées.

$$F_{-1}(x) \quad = \quad F(x+1) - 1 \tag{6}$$

D'après les expériences menées par les auteurs, cette relation est vérifiée, et devient un bonne indicateurs pour les images issues d'appareil photo numériques ou scanners dans des formats avec ou sans perte d'informations.

Cela est tout autre lorsque le plan LSB est modifié de manière aléatoire (par la dissimulation de donnée dans les LSB par exemple). Cette modification réduit la différence entre R_M et S_M en fonction de la longueur du message m. Lorsque l'on arrive à une valeur de 50 % des LSB permutées, ce qui représente la dissimulation d'un message de taille maximum (hypothèse de distribution uniforme des valeurs binaires), on obtient $R_M \cong S_M$. La modification aléatoire des bits de poids faible a l'effet inverse sur R_{-M} et S_{-M}. Dans ce cas, la différence entre les deux valeurs augmente en fonction de la taille du message.

La base de cette méthode de stéganalyse est l'estimation des courbes en fonction du pourcentage de LSB modifié des quatre paramètres R_M, S_M, R_{-M} et S_{-M}. En se basant sur les expériences menées, Fridrich et son équipe en on déduit que les paramètres R_{-M} et S_{-M} peuvent être approximé à l'aide d'une droite. Pour les deux autres paramètres, ils peuvent être approchés par un polynôme du second degré de manière raisonnablement fidèle.

En partant d'une image stéganographiée avec un message d'un longueur inconnue p(en % du nombre de pixels), une première mesure des groupes permet de spécifier les points $R_M(p/2)$, $S_M(p/2)$, $R_{-M}(p/2)$ et $S_{-M}(p/2)$. Le facteur $\frac{1}{2}$ provient de l'hypothèse que le message est uniformément distribué entre les valeurs binaires, ce qui fait que seul la moitié des bits seront modifié en moyenne. En inversant les LSB de l'image en entier, il est possible de calculer les valeurs $R_M(1-p/2)$, $S_M(1-p/2)$, $R_{-M}(1-p/2)$ et $S_{-M}(1-p/2)$. Pour finir, en modifiant de manière aléatoire les LSB de l'image stéganographiée (simulation d'un message de taille maximum), on obtient les points $R_M(1/2)$ et $S_M(1/2)$.

De manière expérimentale, les auteurs ont pu vérifier deux hypothèses :

1. Le point d'intersection des courbes R_M et R_{-M} a la même coordonnée en x que l'intersection de la courbe S_M et S_{-M}.
2. Les courbes R_M et S_M se touche lorsque $m = 50\,\%$, ce qui revient à $R_M(1/2) = S_M(1/2)$.

A partir de ces hypothèses, il est possible de trouver une formule pour le calcul de la longueur p du message dissimulé. En redéfinissant l'échelle de manière à ce que $p/2$ devienne 0 et $1-p/2$ devienne 1, la coordonnée en x de l'intersection devient la racine de la équation du deuxième degré suivante :

$$2(d_1 + d_0)\,x^2 + (d_0 - d_{-1} - d_1 - 3d_0)\,x + d_0 - d_{-0} \quad = \quad 0 \tag{7}$$

Avec les correspondances suivantes :

$$\begin{aligned} d_0 &= R_M(p/2) - S_M(p/2) \\ d_1 &= R_M(1-p/2) - S_M(1-p/2) \\ d_{-0} &= R_{-M}(p/2) - S_{-M}(p/2) \\ d_{-1} &= R_{-M}(1-p/2) - S_{-M}(1-p/2) \end{aligned}$$

On trouve ensuite la longueur p du message en utilisant la racine de l'équation (7) ayant la plus petite valeur absolue, dans l'équation qui suit :

$$p \quad = \quad \frac{x}{(x - \frac{1}{2})} \tag{8}$$

Un petit exemple permettant de visualiser l'impact sur la classification de l'adjonction d'un message à une image peut être donné. Pour ce faire, en se basant sur l'image suivante :

$$I_{base} \quad = \quad \begin{bmatrix} 139 & 144 & 149 & 153 & 155 & 155 & 155 & 155 \\ 144 & 151 & 153 & 156 & 159 & 156 & 156 & 156 \\ 150 & 155 & 160 & 163 & 158 & 156 & 156 & 156 \\ 159 & 161 & 162 & 160 & 160 & 159 & 159 & 159 \\ 159 & 160 & 161 & 162 & 162 & 155 & 155 & 155 \\ 161 & 161 & 161 & 161 & 160 & 157 & 157 & 157 \\ 162 & 162 & 161 & 163 & 162 & 157 & 157 & 157 \\ 162 & 162 & 161 & 161 & 163 & 158 & 158 & 158 \end{bmatrix} \tag{9}$$

Cette image de 8x8 pixels sera utilisé comme image de base, donc non stéganographiée, pour le reste de la démonstration. Sur cette base, des clusters de n pixels adjacents doivent être défini. Ici, n sera égal à 4, ce qui donne, pour la première ligne de l'image, les groupements suivant.

$$G_1 = (139, 144, 149, 153) \text{ et } G_2 = (155, 155, 155, 155)$$

Sur l'ensemble des groupes G définis, on applique la fonction de discrimination (5). Le tableau 21 donne les résultats obtenu sur l'image originale. Ces résultats seront nécessaires pour déterminer la classification de chaque groupes après avoir appliquer la matrice de permutation.

G	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
f(G)	14	0	11	3	13	2	5	1	3	7	0	3	3	5	1	5

TAB. 21 – Résultat de la fonction de discrimination sur l'image de base

Après avoir obtenu ces résultats, il faut maintenant perturber notre image d'origine à l'aide des fonctions réversible, ainsi que de la matrice de permutation $M = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$. L'image résultante aura l'allure suivante :

$$I_{base_{perm}} = \begin{bmatrix} 139 & 145 & 148 & 153 & 155 & 154 & 154 & 155 \\ 144 & 150 & 152 & 156 & 159 & 157 & 157 & 156 \\ 150 & 154 & 161 & 163 & 158 & 157 & 157 & 156 \\ 159 & 160 & 163 & 160 & 160 & 158 & 158 & 159 \\ 159 & 161 & 160 & 162 & 162 & 154 & 154 & 155 \\ 161 & 160 & 160 & 161 & 160 & 156 & 156 & 157 \\ 162 & 163 & 160 & 163 & 162 & 156 & 156 & 157 \\ 162 & 163 & 160 & 161 & 163 & 159 & 159 & 158 \end{bmatrix} \tag{10}$$

Comparativement à (9), seul les éléments centraux sont modifiés en utilisant la fonction F_1. Le tableau 22 compare les valeurs obtenues pour chaque groups, ainsi que leur classification.

G	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$f(G)$	14	0	11	3	13	2	5	1	3	7	0	3	3	5	1	5
$f(F(G))$	14	2	12	3	13	2	7	3	4	9	2	5	7	7	5	5
Classification	U	R	R	U	U	U	R	R	R	R	R	R	R	R	R	U

TAB. 22 – Résultat de la fonction de discrimination et classification

La théorie énoncée plus haut, disant que la perturbation de l'image provoque une augmentation de la fonction de discrimination, est vérifiée. En effet, la majorité des groupes sont défini comme étant réguliers. En dissimulant le message de taille maximum (11), il est possible d'observer l'impact sur une image contenant un message.

$$m = 0000\,0001\,0010\,0011\,0100\,0101\,0110\,0111\,1000\,1001\,1010\,1011\,1100\,1101\,1110\,1111 \tag{11}$$

L'image résultant de la dissimulation du message (11), de manière séquentielle par surécriture du LSB au sein du conteneur est la suivante :

$$I_{Stego} = \begin{bmatrix} 138 & 144 & 148 & 152 & 154 & 154 & 154 & 155 \\ 144 & 150 & 153 & 156 & 158 & 156 & 157 & 157 \\ 150 & 155 & 160 & 162 & 158 & 157 & 156 & 157 \\ 158 & 161 & 163 & 160 & 160 & 159 & 159 & 159 \\ 159 & 160 & 160 & 162 & 163 & 154 & 154 & 155 \\ 161 & 160 & 161 & 160 & 161 & 156 & 157 & 157 \\ 163 & 163 & 160 & 162 & 163 & 157 & 156 & 157 \\ 163 & 163 & 161 & 160 & 163 & 159 & 159 & 159 \end{bmatrix} \quad (12)$$

Comme sur l'image d'origine, il faut appliquer la fonction de discrimination (5), afin de calculer les valeurs de $f(G)$. Ensuite, à l'aide de la même matrice M que précédemment, il faut perturber cette image pour obtenir nos valeurs de $F(f(G))$. Le tableau 23 est une récapitulation des valeurs obtenues au cours de cet exemple.

G	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$f(G_{base})$	14	0	11	3	13	2	5	1	3	7	0	3	3	5	1	5
$f(F(G_{base}))$	14	2	12	3	13	2	7	3	4	9	2	5	7	7	5	5
$Classification_{base}$	U	R	R	U	U	U	R	R	R	R	R	R	R	R	R	U
$f(G_{stego})$	12	1	12	3	12	3	8	1	3	10	3	6	5	8	3	4
$f(F(G_{stego}))$	14	1	12	3	12	3	6	3	3	8	1	6	3	8	3	6
$Classification_{stego}$	R	U	U	U	U	U	S	R	U	S	S	U	S	U	U	R

TAB. 23 – Récapitulation des résultats obtenus

Contrairement à l'image de base, celle contenant un message stéganographié possède plus de groupes de pixel classé comme singulier que ceux étant défini comme régulier. L'image d'exemple étant très petite, la différence n'est que minime. Cette méthode nécessite une plus grande surface pour permettre d'exploiter son potentiel. Les phases suivantes ne seront donc pas traitée avec cette exemple, car le résultat en serait certainement décevant.

Cependant, d'après les expériences menées par les auteurs, l'analyse RS se révèle très efficace pour la détection de contenu stéganographié à l'aide de la méthode de surécriture des LSB.

Les faiblesses de la méthode sont, d'une part quel est inutilisable sur des images fortement bruitées. Se basant sur l'ajout de bruit, elle ne permet pas un seuil de détection suffisant sur des images déjà bruitées. D'autre part, cette méthode est plus efficace sur des images modifiées de manière aléatoire. Si les modification sont ciblées sur une région précise, les résultats de l'analyse risque de ne pas laisser transparaître ces modification. Cependant, en complément à la méthode du χ^2(qui arrive à détecter les modifications séquentielles), une large palette d'algorithme procédant par surécriture du LSB peuvent être détecté.

6.3 Attaque de l'algorithme Outguess

Cette attaque publiée par Jessica Fridrich, Miroslav Goljan et Dorin Hogea [9] permet une estimation de la taille d'un message dissimulé à l'aide de l'algorithme Outguess. Elle ce base sur l'évaluation de données macroscopique au niveau de l'image.

Le principe de base est que l'algorithme Outguess surécrivant les LSB, par conséquent, l'ajout d'un message à une image ayant déjà été stéganographiée vas partiellement annuler les opérations effectuée. Il en résulte donc que l'opération de dissimulation n'aura pas le même impacte sur une image déjà stéganographiée, que sur un conteneur vierge.

Partant de là, une mesure permettant d'exploiter cette particularité à été trouvée. L'algorithme Outguess modifiant de manière aléatoire les coefficients DCT, cela engendrera une augmentation des discontinuités spatial au région frontière des bloc 8x8 utilisé lors de la transformation en DCT.

Cette discontinuité peut être calculée à l'aide de l'équation

$$B = \sum_{i=1}^{\lfloor \frac{M-1}{8} \rfloor} \sum_{j=1}^{N} | g_{8i,j} - g_{8i+1,j} | + \sum_{j=1}^{\lfloor \frac{N-1}{8} \rfloor} \sum_{i=1}^{M} | g_{i,8j} - g_{i,8j+1} |$$

où $g_{i,j}$correspond à la valeur du pixel dans l'image $M \times N$. $\lfloor x \rfloor$ signifiant que seul la partie entière est prise pour résultat.

A partir de là, les étapes suivantes peuvent être effectuée afin de calculer la taille du message caché :

1. Décompresser l'image stéganographiée dans le domaine spatial et en calculer la discontinuité $B_s(0)$
2. En utilisant Outguess, insérer un message de taille maximum. Décompresser l'image, puis calculer la discontinuité $B_s(1)$. Calculer la pente $S = B_s(1) - B_s(0)$
3. Rogner les 4 premières colonnes de pixels de l'image décompressée en 1. Compresser l'image en JPEG en utilisant la même table de quantification que l'image stéganographiée. Décompresser et calculer la discontinuité $B(0)$
4. Insérer un message de taille maximum en utilisant Outguess. Décompresser puis calculer la discontinuité $B(1)$
5. Réinsérer dans l'image précédente un message de taille maximum. Décompresser et calculer la discontinuité $B1(1)$
6. Calculer la longueur du message avec la formule ci-dessous

$$p = \frac{S_0 - S}{S_0 - S_1}$$

Avec $S_0 = B(1) - B(0)$, $S_1 = B1(1) - B(1)$ et $S = B_s(1) - B_s(0)$.

Afin de prouver le bon fonctionnement de leur attaque, un test sur 70 images à été effectuer. Parmi ces images, 24 avaient un message cachée dont la taille variait de 0 à la taille maximum.

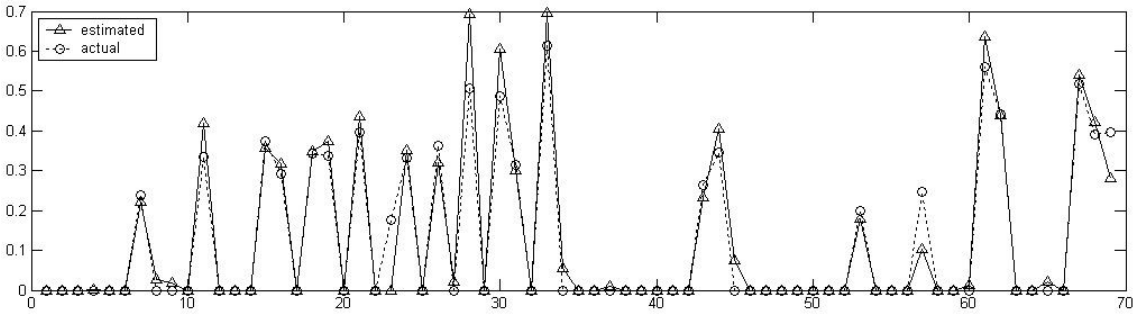

Fig. 42 – Graphique des résultats obtenus [9]

Comme le souligne la figure 42, les résultats obtenu par l'utilisation de cette technique sont très bon. La technique ne soulève pas beaucoup de faux positif.

Cependant, cette technique a des limites. Tout d'abord, si l'image soumise à Outguess est déjà au format JPEG, il la décompressera et la recompressera par la suite. Cela produira donc une double quantification, et la technique énoncée ci-dessus n'est pas efficace sur ces cas particuliers. Pour pouvoir obtenir de bons résultats, il faudra approximer le facteur de qualité Q_c de la première quantification. Le document [9] donne une solution pour estimer ce paramètre.

Le gros inconvénient de cette méthode est qu'elle n'est applicable qu'à un seul algorithme. Pour pouvoir détecter d'autres algorithmes, cette méthode devra être adapté, ou sera inutilisable. De plus, l'impact d'une telle analyse sur une image stéganographié à l'aide d'un autre algorithme est inconnu.

Un exemple simple pour illustrer cette attaque est difficile à donner, étant donnée l'utilisation de l'algorithme lui même dans l'analyse. Cette attaque ne sera donc pas illustrée.

7 Techniques Universelles

Les techniques universelles, aussi appelée "blind steganography", sont essentiellement basées sur des architectures de type réseaux de neurones. Le principe étant de soumettre une base d'entraînements aussi diversifiée que possible, afin qu'il puisse détecter un large panel de technique stéganographique.

Le point le plus difficile étant de trouvé les caractéristiques du conteneur permettant de différentier un contenu propre, d'un autre étant stéganographié.

On s'accorde généralement à dire que les techniques de stéganalyse universelles permette une détection d'un plus large panel de contenu stéganographié. Cependant, contrairement au techniques plus spécifique, les résultats émanant de ce type d'architecture est de type binaire (stéganographié/non stéganographié). Aucune estimation de taille n'est possible. Il sont aussi moins efficace comparativement à une technique spécifique sur un algorithmes déterminé.

Quatrième partie

Conclusion

La stéganographie représente un gros challenge dans le monde de la sécurité informatique. Sous l'impulsion des milieux académiques, des techniques de plus en plus sophistiquée sont développées. En parallèle, des outils de stéganalyse de plus en plus évolué apparaissent. La plupart sont très difficile à appliquer en dehors de leur contexte mathématiques. Cependant, la question la plus importante à se poser reste de savoir si, à part l'informaticien du dimanche voulant absolument tester les dernier gadget, la stéganographie est réellement utilisée à des fins criminelles. Ce document ne permet pas de répondre à cette question.

Un autre élément à souligner est la forte recrudescence de documentation techniques concernant la stéganographie sur images numériques. Près de 90 % des documents trouvé s'attache à décrire les possibilités offertes par ce support. D'un autre côté, très peu de document traite de l'audio, ainsi que des méthodes de stéganalyse applicable à l'audio. Le prolongement de ce travail pourrait aboutir sur la tentative d'adaptation, du moins l'analyse de la faisabilité, des méthodes de stéganalyse image au contenu audio.

Au cours des diverses lectures, on se rend compte que la stéganographie constitue un éternelle recommencement.

Ljupce Nikolov