Stéganographie

Gabriel Champiat

9 mai 2020

Table des matières

1 Introduction

Dans le cadre de nos cours à l'Ecole Nationale Supérieure d'Ingénieurs de Bretagne Sud, nous avons eu une introduction à la Stéganographie. Afin de nous exercer dans ce domaine, nous avons dû implementer un programme permettant de mettre en place une LSB.

L'objectif d'une LSB est d'insérer des données dans les bits de poids faible de chaque octet composant l'image. En effet, avec une insertion de ce type, nous ne pouvons pas faire la distinction à l'oeil nu.

Dans un premier temps, nous détaillerons la conception de notre programme en expliquant notamment l'insertion des données et la récupération des données. Puis nous continuerons par expliquer comment nous avons mis en place la déctection d'une LSB dans une image. Puis nous finirons par la présentation des courbes ROC obtenues avec notre fonction de détéction.

1.1 Acronymes et Définitions

Acronymes

LSB Last Significant Bit

RGB Red Green Blue

ROC Receiver Operating Characteristics

Définitions

stégo-cover Image contenant de la donnée via une technique d'insertion telle que le LSB.

stégo-médium Image ayant servi comme image de base pour l'insertion d'un message.

2 Technique LSB

2.1 Architecture

Dans un premiers temps, le programme a été ecrit en Golang, il faudra donc, pour l'utiliser passer par une phase de compilation. Nous avons choisi ce langage, pour plusieurs raisons. La première raison est que ce langage nous permet de compiler le même code pour différentes architectures(e.g. Linux, Windows, MacOS) ce qui est très plaisant. De plus, la compilation se fait de manière statique, ce qui nous permet d'avoir un programme prêt à l'utilisation, sans l'installation de dépédences. Et enfin car ce TP nous a permis de mettre en oeuvre nos conaissances théoriquse sur ce langage.

Afin de rendre notre conception évoluable, nous sommes partis sur l'utilisation du patron de conception "Décorateur". En effet grâce à ce patron de conception, nous pourrons par la suite rajouter des fonctionnalités à notre programme Vous trouverez en [1] une explication claire du fonctionnement de ce patron de conception.

Pour des raisons évidentes de lisibitité et de maintenabilité nous avons divisé notre programme en plusieurs packages. Ci-dessous une explication rapide de l'utilité de chaque package.

image Package qui défini toutes les méthodes et structures permettant d'intéragir avec une image.

lsb Le package lsb permet de définir toutes les méthodes et structures permettant l'insertion, la récupération et la détéction de données dans une image.

utils Référence toutes les méthodes communes entre tous les packages. En effet, les méthodes ecritent dans ce package sont des méthodes très génériques.

cmd Ce package défini le fonctionnement de notre application quand celle-ci est lancée depuis une ligne de commande.

Vous trouverez en annexe de ce document toutes les intructions pour compiler et utiliser notre programme

2.2 Insertion

La première étape de l'insertion est de vérifier que le stego-médium possède assez d'octets pour acceuillir la taille du message. Pour cela, il nous suffit de multiplier par 8 la taille du message pour avoir sa taille en bits, puis de comparer cette valeur avec le nombre pixels disponible dans l'image. Cette étape permet de calculer le taux stéganographique.

Dans notre cas nous rajoutons au message d'origine une entête de 8 octets permettant de définir la taille du message, ce qui nous sera utile lors de récupération du message.

Pour le chiffrement des données, nous avons mis en place l'algorithme de chiffrement RC4. Nous avons choisi un algorithme de chiffrement par flux pour la simple et bonne raison d'éviter de rajouter de la taille à notre message. Par exemple l'utilisation du chiffrement AES va augmenter la taille du message et par conséquence augmenter le taux stéganographique, ce qui a pour conséquence d'augmenter la probabilité de détéction.

Mainteant pour l'insertion des données, il suffit de parcourir tous les octets de l'image afin d'insérer chaque bit du message. Pour parcourir les octets le programme fonctionne de deux manières différentes. La première lorsque l'utilisateur ne saisi pas de graine, alors le programme va lire les octets les uns à la suites des autres. Si l'utilisateur saisi une graine, alors le programme va simplement tirer des nombres "aléatoire" depuis cette graine pour choisir l'emplacement de l'octet. Afin de simplifier l'utilisation du programme la graine devra être sous forme de chaine de caractère qui sera par la suite transformée en entier. Comme dit précédemment, nous gérons cette différence de comportement grâce au patron de conception Décorateur.

Ensuite pour l'insertion, nous réalisons la technique de LSB Replacement, c'est-à-dire que peu importe la valeur du bit de poids faible de l'octet, nous allons le remplacer la par valeur du bit que nous souhaitons insérer.

Ci-dessous un exemple d'insertion

```
1 go-lsb -insert image_src.bmp image_dst.bmp text.txt
```

Pour vérifier que l'algorithme fonctionne bien, ci-dessous un stégo-cover et un stégo-médium. Seriez-vous capable de savoir quelle image est le stégo-médium et quelle image est le stégo-cover ?

FIGURE 1 – Stégo-medium vs Stégo-cover

2.3 Récupération

Maintenant que nous avons vu comment notre programme insére les données, nous allons voir comment celui-ci les récupérer. La première étape est de récupérer la taille du message qui est stockée dans le header.

Comme pour l'insertion, pour parcourir les octets de l'image deux choix sont possibles soit l'utilisateur n'a pas saisi de graine, dans ce cas là, le programme va lire à la suite. Sinon comme pour l'insertion via une graine sous forme de chaine de caractère, le programme va lire des octets de manière "aléatoire".

Une fois la taille de message récupérée, il suffit, à l'aide d'une boucle de récupérer tous les octets nécessaires, à l'aide du même fonctionnement décrit juste avant. Une fois que nous avons le message, celui-est déchiffré si l'utilsateur a saisie une clé.

Dans tous les cas, le message est affiché sur la sortie standard de l'utilisateur.

2.4 Detéction

Comme vous avez pu le voir lors de l'insertion, le LSB est quasi indétectable à l'oeil nu, mais ce n'est pas pour autant que nous ne pouvons pas le detecter. En effet, il est possible de detecter l'insertion de données grâce à une étude statistique du nombre de bits de poid faible avec la valeur 0 ou 1.

Attention, comme tous les tests stastiques, celui-ci dépend grandement de la population étudiée. Ici nos tests dépendrons uniquement des stégo-medium et du taux sténaugraphique dans les stégo-cover.

Pour réaliser cette détection nous allons utiliser l'algorithme Sample Pair Analysis (SPA). Dans notre implementation de l'algorithme, nous allons comparer pour tous les pixels (i.e 3 octets) de l'image leurs valeurs RGB avec le pixel voisin. En effet, cette détection se base sur le fait que stastiquement deux pixels voisins dans une image auront des valeurs semblables.

Vous trouvez en annexe de ce document le pseudo-code de l'algorithme.

3 Courbes ROC

Afin de mettre à l'épreuve notre fonction de détéction, nous allons créer des courbes ROC (Receiver Operating Characteristics) à partir des résultats de celle-ci. En effet ces courbes offrent à la fois une vision graphique et une mesure pertinente de la performance d'un classifieur. Notre fonction de détéction est bien un classifieur, car celle-ci indique si oui ou non(i.e. réponse binaire) l'image est un stégo-cover.

Pour tracer les courbes nous placerons en absicisse les faux positifs et en ordonnées les vrais positifs.

Maintenant, nous allons nous procurer notre jeu de données afin de réaliser des tests. Pour cela, nous allons nous utiliser le site suivant https://picsum.photos/. Notre jeu de données sera composé de 100 images d'une taille de 200x300 pixels.

Maintenant que nous avons notre dataset, nous allons créer nos courbes avec un simple programme python. Afin d'avoir le résultat de notre fonction de détéction, nous allons utiliser notre binaire Golang.

Le fonctionnement du programme python est le suivant, nous allons parcourir notre dataset d'image (dans notre cas 100 images) et insérer de la donnée dans ces images avec des taux stéganographiques différents. Dans un premiers temps, nous allons expliquer la procédure, pour cela nous utiliserons un taux stéganographique de 5%. Maintenant que nous avons les stégo-cover et les stégo-médium, nous allons pour chaque catégorie d'image exécuter notre fonction de détection et stocker les résultats.

Ensuite on fait varier le taux à partir duquel l'image est considérée comme stégo-cover. Cette variation a un pas de 0.01 et est bornée entre 0 et 1. Pour chaque variation, nous allons calculer le nombre de vrai positif et le nombre de faux positif. Nous appellons un vrai positif quand la détéction indique de la données dans un stégo-médium et inversement, un faux positif quand la fonction indique de la données dans un stégo-medium.

Pour finir, nous divisions la valeur de vrai positif et de faux positif par le nombre d'image correspondant à chaque catégorie, c'est-à-dire au nombre d'images considérees comme stego-médium et stégo-cover. Dans notre cas il s'agit de cent images pour chaque catégorie.

Une fois que nous avons réalisé tous les calculs, il suffit simplement d'afficher les résultats sur une courbe.

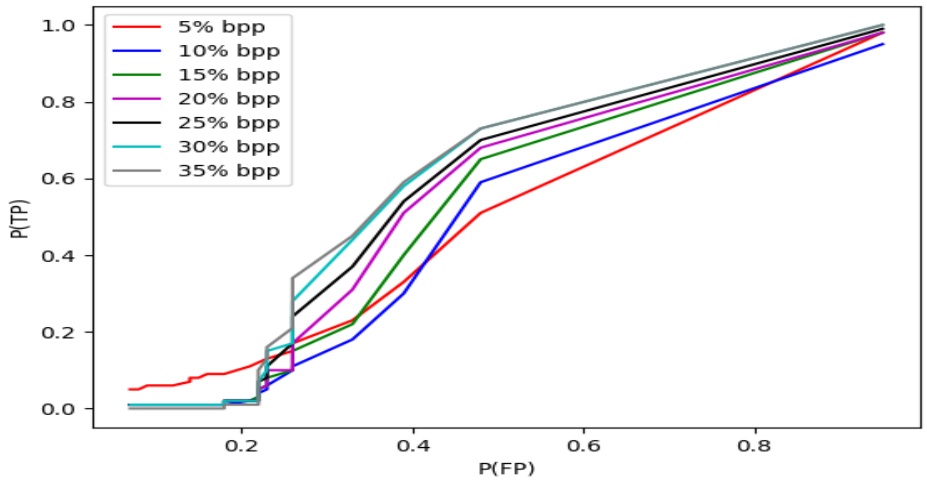

FIGURE 2 – Courbe ROC

Afin de comparer nous avons réalisé les courbes ROC avec des insértions aléatoires

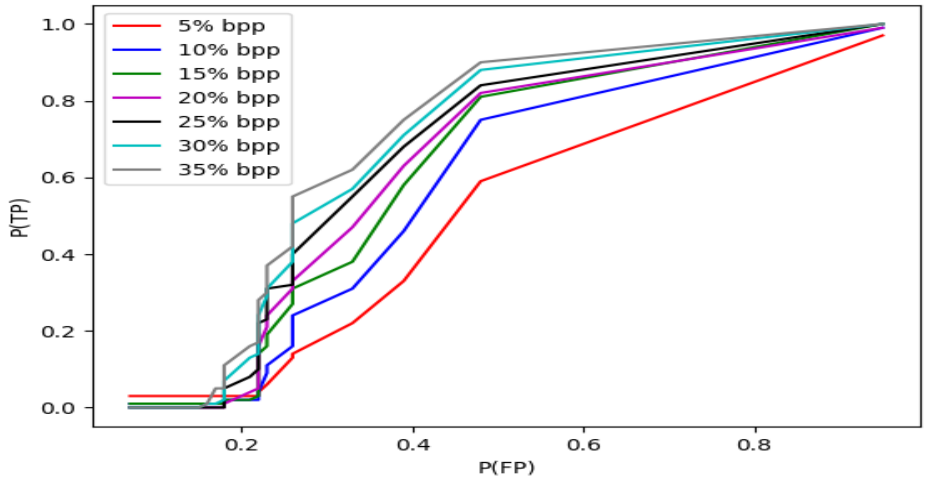

FIGURE 3 – Courbe ROC avec aléatoire

Comme vous pouvez le voir les courbes sont légérement différentes. On remarque que notre fonction de détéction fonctionne un peu mieux sur des données avec insertions aléatoire. Cela se remarque par le fait que si nous regardons la valeur en Y pour un X donné, alors la valeur de Y sera supérieure sur le second graphique.

4 Conclusion

Ce travail pratique fut très intéressant et très formateur. En effet, dans un premier temps, celui-ci nous a permis d'appréhender le langage Golang. Mais ce n'est pas tout la technique de LSB est une technique de dissumlation de données que nous avions rapidement vu, mais sans jamais prendre le temps nécessaire à sa bonne comprehension. L'implementation de notre propre version du LSB nous a permit d'assimiler entièrement son fonctionnement.

5 Annexes

5.1 Compilation

Pour compiler le programme veuillez-vous assurer dans un premier temps que vous avez bien Go d'installé sur votre ordinateur et que la variable globale GOPATH possède une valeur. Vous pouvez voir les valeurs des variables Golang avec la commande suivante.

```
1 go env
```

Ensuite, il faut déplacer le contenu de l'archive dans le répertoire **$GOPATH/src/github.com/gachampiat**. Il faut maintenant télécharger les dépendances de notre programme pour cela tapez la commande suivante

```
1 go get golang.org/x/image/bmp
```

Maintenant que vous avez toutes les dépendences vous être prêt à utiliser notre programme, pour cela, vous devez le compiler

```
1 go build go-lsb.go
```

Après cette commande, un binaire nommé **go-lsb** devrait être créé dans votre répertoire courrant.

5.2 Utilisation

Pour avoir une aide sur l'utilisation du programme vous pouvez simplement l'appeller sans argument, une aide d'utilisation devrait apparaître.

```
1 go-lsb
```

Ci-dessous des exemples d'utilisation pour que vous puissiez bien comprendre son fonctionnement.

Insertion

Ici, nous avons une insertion simple dans une image. Le texte à insérer se trouve dans le fichier text.txt.

```
1 go-lsb -insert image_src.bmp image_dst.bmp text.txt
```

Si maintenant nous voulons chiffrer les données insérées il suffit de rajouter l'option

key, comme ci-dessous.

```
1 go-lsb -key MYKEY -insert image_src.bmp image_dst.bmp text
    .txt
```

Enfin si nous voulons insérer les données de manière "aléatoire"

```
1 go-lsb -key MYKEY -seed THISISMYSEED -insert image_src.bmp
    image_dst.bmp text.txt
```

Récupération

Après avoir vu comment insérer les données nous allons voir comment les récupérer. Dans un premier temps, nous allons efféctuer une récupération simple.

```
1 go-lsb -retrive image_dst.bmp
```

Le contenu du fichier text.txt sera alors affiché sur la sortie standard. Ensuite comme pour l'insertion, si vous voulez dechiffrer le text, il suffit de rajouter l'option key.

```
1 go-lsb -key MYKEY -retrive image_dst.bmp
```

Et enfin pour finir si vous voulez récupérer le contenu qui a été inséré aléatoirement, il suffit de rajouter l'option seed.

```
1 go-lsb -key MYKEY -seed THISISMYSEED -retrive image_dst.
    bmp
```

Détéction

Pour lancer la détéction sur une image, il suffit de taper la commande suivante

```
1 go-lsb -detect image_dst.bmp
```

5.3 Documentation

Afin d'avoir la documentation vous pouvez utiliser la commande suivante

```
1 go doc package_name
```

Par exemple pour voir la documentation du package lsb il suffit de taper le commande suivante

```
1 go doc -all ./lsb
```

5.4 SPA

```
VARIABLES
     uint : x, y, k
     uint8 : r, s

POUR i ALLANT_DE 0 A taille de l image {PAR_PAS_DE 1}
    POUR j ALLANT_DE 0 A largeur de l image {PAR_PAS_DE 1}
        POUR c ALLANT_DE 0 A 3 {PAR_PAS_DE 1}
            r := pixels[i][j][c]
            s := pixels[i][j+1][c]
            if (s%2 == 0 && r < s) || (s%2 == 1 && r > s)
            {
                x++
            }
            if (s%2 == 0 && r > s) || (s%2 == 1 && r < s)
            {
                y++
            }
            if math.Round(float64(s)/2) == math.Round(
               float64(r)/2)
            {
                k++
            }
        FIN_POUR
    FIN_POUR
FIN_POUR
```

Références

[1] *GOPROD • GOod PRactice in Object oriented Design.* URL : http://www.goprod.bouhours.net/.