esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

Enseignant
Nicolas Bodin

MS-SIS

Introduction à la stéganographie

Résumé

Après le point de cours qui vous a été fourni, vous êtes maintenant prêts à passer à la pratique. Ce TP se focalise sur l'insertion d'un message dans une image de type JPEG. Après avoir analysé le code fourni permettant de manipuler la librairie JPEG, vous devrez créer un programme d'insertion LSB et de vérifier expérimentalement ses effets sur les images JPEG.

Le rendu se fera sous la forme d'un court rapport, ainsi que d'un code source, le tout dans une archive ZIP nommée **NOM_Prenom.zip**.

1 Le format JPEG

L'objectif de cette section est de comprendre le fonctionnement du format JPEG. À l'aide du code source fourni, vous allez manipuler les coefficients d'une image JPEG pour voir l'effet des différentes étapes de la compression, en particulier l'application de la DCT.

1.1 Prise en main du code source

Le code source qui vous est fourni est prévu pour compiler et fonctionner sous Linux et Windows. Il permet de copier une image au format JPEG en ayant accès à ses coefficient DCT quantifiés. Dans un premier temps, vérifiez que vous pouvez compiler et exécuter ce code qui réalise simplement la copie d'une image JPEG.

Le code comporte la librairie officielle JPEG dans le dossier **JPEGlib**. Il est inutile de modifier le contenu de ce dossier. La librairie de manipulation d'une image JPEG s'articule autour des fichiers *jpeg_manip.c/.h*. Pour toute la suite de cette section, on définit les variables suivantes dans le fichier *main.c*

```
20 int return_value;         // Variable temporaire
21 JPEGimg *img = NULL;      // Pointeur sur l'image JPEG
22 DCTpos position = { 0 }; // Position d'un coefficient DCT
```

Listing 1: Définitions des variables du `main` dans le fichier *main.c*.

La première étape consiste à ouvrir une image JPEG et stocker ses coefficients DCT. Cette étape est réalisée dans l'extrait de code suivant.

```
41 img = jpeg_read(argv[1]);
```

Listing 2: Lecture d'une image JPEG passée en argument de la ligne de commande dans le fichier *main.c*.

Comme l'indique le listing 1, l'image `img` est donc un pointeur sur une structure `JPEGimg` définie dans le fichier *jpeg_manip.h*. Cette dernière est au centre de toutes les fonctions et est décrite dans le listing suivant.

```
21 typedef struct JPEGimg_s
22 {
23   JBLOCKARRAY * dctCoeffs;
24   struct jpeg_decompress_struct cinfo;
25   jvirt_barray_ptr * virtCoeffs;
26 } JPEGimg;
```

Listing 3: Définition de la structure `JPEGimg` dans le fichier *jpeg_manip.h*.

Le champ `dctCoeffs` donne accès à un tableau à 4 dimensions contenant les coefficients DCT quantifiés de l'image. La première dimension représente la composante, les 2 et 3ème dimensions donnent respectivement le numéro de ligne et de colonne d'un bloc DCT. La dernière indique quel coefficient choisir parmi les 64 d'un bloc. Par exemple, `dctCoeffs[1][10][3][27]` nous donne la valeur du 27ème coefficient DCT du bloc ligne n°10, colonne 3, de la composante 1 (chrominance bleue).

Le champ `cinfo` est une structure donnant des informations générales sur l'image. Nous utiliserons particulièrement les champs `num_components` et `comp_info`. Le premier indique le nombre de composantes présentes dans l'image (en général, 1 pour une image en niveaux de gris, 3 pour une image couleur). Le deuxième est un tableau de taille `num_components`. Chaque case est une structure donnant, entre autres, le nombre de blocs DCT par ligne et par colonne dans les champs `height_in_blocks` et `width_in_blocks`. L'extrait de code suivant vous indique un moyen d'accéder à ces valeurs.

```
46 printf("Number of components: %d\n", img->cinfo.num_components);
47 for (i = 0; i<img->cinfo.num_components; i++)
48 {
49   printf("Component %d:\n   %d lines, %d columns\n\n", i,
     ↪  img->cinfo.comp_info[i].height_in_blocks,
     ↪  img->cinfo.comp_info[i].width_in_blocks);
50 }
```

Listing 4: Extrait de code du fichier *main.c*

Parce que les coefficients DCT d'une image JPEG sont donnés sous la forme d'un tableau à 4 dimensions, une structure `DCTpos` a été définie dans le fichier *jpeg_manip.h*. Elle regroupe 4 entiers donnant la composante, la ligne et la colonne d'un bloc DCT, et enfin la position du coefficient au sein de ce bloc.

Le listing 5 vous indique l'utilisation des fonctions `getDCTpos()` et `getDCTcoeffValue()`. La fonction `getDCTpos()` définie dans *jpeg_manip.c* permet d'associer une position dans une image à partir d'un entier. Par exemple, à la valeur 0 est associé la position de coefficient DCT `(0,0,0,0)`, à la valeur 1 la position `(0,0,0,1)`, à la valeur 64 la position `(0,0,1,0)` ... La fonction `getDCTcoeffValue()` permet de récupérer la valeur d'un coefficient étant donnée sa position dans l'image (en `DCTpos`).

Enfin l'écriture de l'image dans un nouveau fichier et la libération de la mémoire s'effectuent à l'aide des lignes de code données dans le listing 6.

```
46 getDCTpos(img->dctCoeffs, &(img->cinfo), 0, &position);
47 getDCTcoeffValue(img->dctCoeffs, &(img->cinfo), &position,
    ↪ &return_value);
48 printf("First coefficient (comp 0, lin 0, col 0, coeff 0): %d\n",
    ↪ return_value);
```

Listing 5: Utilisation des fonctions `getDCTpos()` et `getDCTcoeffValue()` dans le fichier *main.c*

```
46 return_value = jpeg_write_from_coeffs(argv[2], img);
47 if (return_value == EXIT_SUCCESS)
48   printf("\nImage written in %s\n", argv[2]);
49 free_jpeg_img(img);
```

Listing 6: Écriture de l'image et libération mémoire dans le fichier *main.c*

En pratique, les coefficients DCT sont stockées sur 16 bits car la DCT implique des valeurs de coefficients entre -4080 et 4080 pour des blocs de taille 8×8 (ce qui est le plus fréquent). Nous allons maintenant modifier les coefficients du premier bloc DCT afin de voir l'effet cette transformée en cosinus discret.

Exercice 1. Prise en main du basecode.

1) Une fois l'image ouverte, donnez la valeur 1000 au coefficient n°1 de la première composante du premier bloc DCT de l'image. Observez l'image produite et ajoutez une capture de la zone modifiée dans votre rapport.
2) Faites de même avec les coefficients n°7, 8 et 63.
3) Comparez les résultats de la modification du premier coefficient sur chacune des 3 composantes. Expliquez ce résultat.
4) Focalisez-vous à nouveau sur la composante n°1. De nouveau modifiez le coefficient n°0 de $+1$, puis de même avec le n°1, le n°7 et le n°63. Quelle est la modification la plus visible ?

Cet exercice vous a donc permis de comprendre que la position d'un coefficient DCT était directement lié à la fréquence que portait le coefficient. L'amplitude de la modification joue également un rôle important sur l'aperçu final du bloc. Rappelons que notre objectif étant d'insérer un message de la façon la plus discrète possible, vous devriez déjà avoir une idée des coefficients à éviter de modifier.

2 Insertion dans une image JPEG

Vous allez donc maintenant coder une insertion de type LSB replacing dans une image JPEG. Pour rappel, le LSB replacing consiste à remplacer le bit de poids faible d'un coefficient DCT si il ne correspond au bit du message à insérer. Parce que le but est d'étudier les effets d'une telle insertion, vous n'allez pas insérer un vrai fichier dans l'image. Vous allez simplement réaliser une simulation d'insertion à un taux prédéfini.

La simulation consiste à insérer un message aléatoire dans l'image. Pour cela, le plus simple est de parcourir l'ensemble des coefficient DCT de l'image. Pour chacun d'entre eux, un bit aléatoire est généré. Si ce dernier est à 0, pas de modification, sinon, on modifie

le LSB du coefficient DCT.

Exercice 2. Simulation d'insertion de type LSB replacing

Votre fonction prendra en paramètre un pointeur sur une image JPEG déjà ouverte, un taux d'insertion entre 0 et 1 et le chemin de la stégo image.

1) Rédigez un code permettant de calculer le nombre de coefficients DCT d'une image JPEG. Utilisez pour cela le champ `cinfo` du pointeur `img` comme montré dans le listing 4.
2) Parcourez les coefficients DCT et modifiez les selon l'algorithme présenté ci-dessus.
3) Lancez votre code avec un taux d'insertion de 0,5 bit par coefficient DCT.
4) Installez sur vos postes l'utilitaire Image Magick et lancez la commande `compare --metric=PSNR <cover> <stego>` afin de voir quelles zones de l'image ont été modifiées. Ajoutez cette image à votre rapport.

Le dernier exercice a permis de montrer des différences flagrantes entre la cover et la stégo image. Au delà des différences visuelles, nous allons étudier des propriétés statistiques simples permettant de mettre en évidence l'utilisation de la stéganographie sur une image. Il s'agit de la *stéganalyse* dont l'objectif est simplement de détecter la présence d'une communication secrète et non de récupérer le message.

Le principe d'une stéganalyse est toujours le même et se divise en 4 étapes. La première consiste à récupérer des propriétés statistiques de l'image, appelées aussi *caractéristiques*. L'objectif est de se focaliser sur des propriétés qui risquent d'être altérées lors d'une insertion.

La deuxième étape consiste à apprendre ce qu'est une cover et ce qu'est une stégo image à l'aide d'un apprentissage supervisé. L'idée est de donner à une machine un ensemble de caractéristiques de cover et de stégo afin que cette machine créée une frontière entre ces deux types de données. Notons que la plupart du temps, un apprentissage est réalisé pour un taux d'insertion particulier.

La troisième étape consiste à vérifier l'exactitude des décisions prises à l'issue de l'apprentissage. Pour cela, on donne à nouveau à notre oracle un ensemble de cover et de stégo (à nouveau à taux d'insertion fixé) et on note la décision prise par ce dernier pour chaque élément. Des scores sont alors calculés et permettent de vérifier l'efficacité de l'oracle. Tant que ces derniers ne sont pas satisfaisant, on retourne à la première étape.

Enfin la dernière étape consiste à tester les objets de la communication suspectée. Pour chacun d'entre eux, l'oracle décide si oui ou non une communication secrète existe. Lorsqu'une alerte est remontée, des recherches approfondies sont menées (forensic) afin de valider ou invalider le résultats de l'oracle. Dans tous les cas, rien n'est certain tant que le message n'est pas récupéré ! Et il s'agit d'une tâche encore plus difficile...

Exercice 3. Stéganalyse simple

Les caractéristiques extraites pour cette stéganalyse sont basées sur l'histogramme d'un image. Il s'agit simplement d'un tableau comptant le nombre de coefficients DCT dont la valeur est de 0, de 1, de 2 etc. En effet, le LSB replacing est susceptible de modifier suffisamment l'histogramme pour y mener une stéganalyse viable.

1) Rédigez une fonction permettant de récupérer l'histogramme d'une image JPEG pour des valeurs de coefficient entre −100 et +100. Exportez vos résultats dans un fichier CSV afin de les tracer avec le logiciel `gnuplot`.
2) Comparez l'histogramme d'une cover image et de son stégo associé à une taux d'insertion de 1. Ajoutez le deux histogrammes à votre rapport et commentez les résultats.

3) En rappelant que le LSB replacing modifie la valeur du bit de poids faible lorsque ce dernier est différent de celui du message, expliquez la forme de l'histogramme de la stégo image au taux 1.
4) Sans nécessairement le coder, donnez un processus permettant la détection du LSB replacing.
5) En vous aidant de la question 4 de l'exercice 1, donnez quelques contre-mesures afin de renforcer la sécurité l'insertion.

3 Conclusion

Cet atelier vous a permis de découvrir les méthodes basiques de la stéganographie. Actuellement, les recherches se concentrent toujours l'utilisation des images JPEG mais aussi non compressées. La stéganographie de type LSB replacing est révolue depuis 1998, date d'apparition de la stéganographie moderne. Dans ces nouveaux type de stéganographie, un coefficient ne porte plus l'information d'un bit de message. Une seule modification permet l'insertion de plusieurs bits de message.

De plus, les modifications ne se font plus séquentiellement, mais se concentrent sur des zones favorables. Il s'agit de zones à forts contrastes (donc à hautes fréquences) dans lesquelles la modification d'un coefficient est particulièrement difficile à détecter. De plus, l'ordre dans lequel ces coefficients sont choisis est déterminé par l'usage d'une clé stéganographique. Cette dernière permet de créer une permutation sur l'ensemble des coefficients afin de répartir les modifications sur l'ensemble de l'image et non sur les premières valeurs.

Pour ceux d'entre vous qui sont particulièrement intéressés par ce domaine, je vous encourage à feuilleter le livre référence en stéganographie. Rédigé par Jessica Fridrich en 2010, *Steganography in digital media* permet d'aborder la stéganographie comme vous venez de le faire, mais aussi de vous initier au *matrix embedding*, aux *wet papers* etc. Son site internet regorge également de codes source et d'articles de recherche. Le niveau est très technique, mais si vous êtes motivés, rien ne saurait vous arrêter !