2019

Rapport Mini Projet

Compression et Stéganographie

Une application à distance pour cacher un fichier compressé dans une image au niveau de gris.

Module : Sécurité Informatique
Filière : Master SIDI
Département Informatique

Responsable :
Prof. Fatima AMOUNAS

Réalisé par : Hassan ZEKKOURI
5/26/2019

Table de matières

Introduction et objectif

Nous tenons à remercier la Prof. AMOUNAS Fatima pour nous avoir donné l'occasion de travailler sur un tel projet dans le domaine de stéganographie à coté de domaine de la compression des données. Alors l'objectif de projet consiste, dans un premier temps, à compresser un fichier quelconque et puis, dans un second temps, à en dissimuler dans un support. L'application implémentée est une application à distance RMI dotée de quatre fonctions principales:

- Compress ;
- Encode ;
- Decode ;
- Decompress.

L'application est développée sous l'environnement NetBeans IDE 8.2 !

I. Partie A : Algorithme LZW

L'algorithme **LZW** (pour *Lempel-Ziv-Welch*) est un algorithme de compression de données sans perte. Il s'agit d'une amélioration de l'algorithme LZ78 inventé par Abraham Lempel et Jacob Ziv en 1978. LZW fut créé en 1984 par Terry Welch, d'où son nom.

1) La compression

L'idée générale est de tirer parti des motifs répètes dans le fichier à compresser. Un tel motif se voit attribuer un code particulier. Par la suite, toute nouvelle apparition de ce motif est remplacée par le code correspondant ce qui peut mener à de substantielles économies de place. On se retrouve alors avec un dictionnaire entre motifs et codes. Initialement le dictionnaire associe à chaque caractère son code ASCII.

```
FONCTION LZW_Compresser(Texte, dictionnaire)
    w ← ""
    TANT QUE (il reste des caractères à lire dans Texte) FAIRE
       c ← Lire(Texte)
       p ← Concaténer (w, c)
       SI Existe (p, dictionnaire) ALORS
          w ← p
       SINON
          Ajouter (p, dictionnaire)
          Écrire code[w]
          w ← c
    FIN TANT QUE
FIN
```

2) Décompression

Cette partie consiste à renverser le processus : on reçoit une liste de codes, et il s'agit de retrouver la chaîne d'origine. De nouveau, on dispose d'un dictionnaire, initialement rempli avec les caractères et leurs codes ASCII.

```
FONCTION LZW_Décompresser(Code, dictionnaire)
     n ← |Code|
     v ← Lire(Code)
     Écrire dictionnaire[v]
     w ← char(v)
     POUR i ALLANT DE 2 à n FAIRE
        v ← Lire(Code)
        SI Existe (v, dictionnaire) ALORS
          entrée ← dictionnaire[v]
        SINON entrée ← Concaténer (w, w[0])
        Écrire entrée
        Ajouter(Concaténer(w, entrée[0]), dictionnaire)
        w ← entrée
     FIN POUR
FIN
```

Les deux algorithmes sont disponibles sur Wikipédia :

Veuillez suivre ce lien : LZW

II. Partie B : Stéganographie

La stéganographie consiste à cacher un message « secret » dans un contenu « support ». Le message secret doit être invisible aux observateurs du contenu support, et son existence même doit idéalement être indétectable par un adversaire qui chercherait à décider si le contenu contient ou pas un message caché (on parle alors de stéganalyse).

Le message caché doit bien sûr pouvoir être retrouver par ceux qui connaissent son existence et savent décoder cette information. Ce travail s'intéresse à une des première technique stéganographie, dite Least Significat Bit (LSB, « bit de poids faibles »), qui permet de cacher un message secret quelconque (de texte par exemple) dans une image dans une image numérique. Ici, on travaille avec une image au niveau de gris sur 8bits.

Pour la suite on a la structure suivant :

Le code est commenté à fin de spécifier le rôle de chaque fonction et de chaque classe !

- La compression/décompression est regroupé dans une classe LZW ;
- Le Codage/Décodage est regroupé dans une classe Steganography ;
- Une classe pour la technique LSB appelée StegaLSB ;
- Une classe pour l'image au niveau de gris PGM type P2 fichier ASCII ;
  Image PGM : Portanble GrayMap

1) Codage :

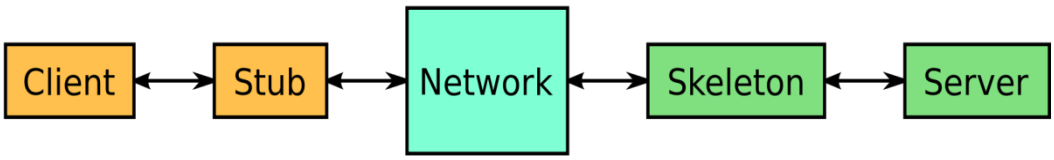
L'algorithme de codage :

```
1. Définir le n : nombre de bits par pixels ;
2. Charger l'image ;
3. Ouvrir le fichier à cacher contenant le message secret ;
4. Calculer la taille du fichier à cacher ;
5. Si cette taille est compatible avec la taille de l'image ;
    5.1. Créer une nouvelle image de taille Nlignes x Ncolonnes
         pixels ;
    5.2. Recopier l'image source dans cette nouvelle image ;
    5.3. Encoder le nom du fichier à cacher dans les 12x8/n premiers
         pixels de l'image ;
    5.4. Encoder la taille du fichier à cacher dans les 4x8/n pixels
         suivants ;
    5.5. Tant que le fichier à cacher n'est pas complètement lu :
        5.5.1. Lire en binaire un octet du fichier à cacher ;
        5.5.2. Encoder cet octet dans 8/n pixels de l'image ;
    5.6. Sauvegarder l'image dans un fichier.
```

Pour la fonction decode, il suffit d'inverser ce processus de codage à partir de l'étape 5.3. Et de stocker dans un fichier final qui va contenir le message secret!

III. Implémentation

1) Application à distance RMI :



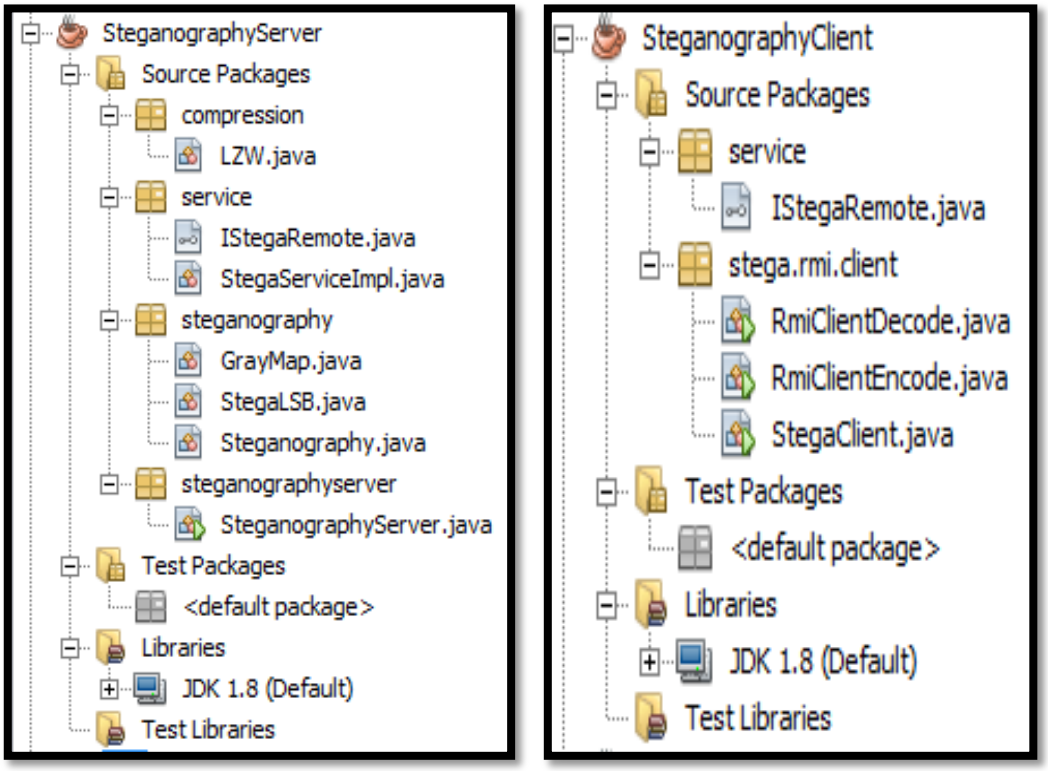
Remote method invocation, plus connu sous l'acronyme **RMI** est une interface de programmation(API) pour le langage Java qui permet d'appeler des méthodes distantes, sur le principe des ORB. L'utilisation de cette API nécessite l'emploi d'un registre RMI sur la machine distante hébergeant ces objets que l'on désire appeler au niveau duquel ils ont été enregistrés.

Cette bibliothèque, qui se trouve en standard dans Java SE, permet la communication via le protocole HTTP (ou IIOP, depuis la version 1.3 du JDK) entre des objets Java éloignés physiquement les uns des autres, autrement dit s'exécutant sur des machines virtuelles java distinctes. RMI facilite le développement des applications distribuées en masquant au développeur la communication client / serveur.

Source : https://fr.wikipedia.org/wiki/Remote_method_invocation

2) Architecture de l'application :

a) Serveur Stéganographie et clients:



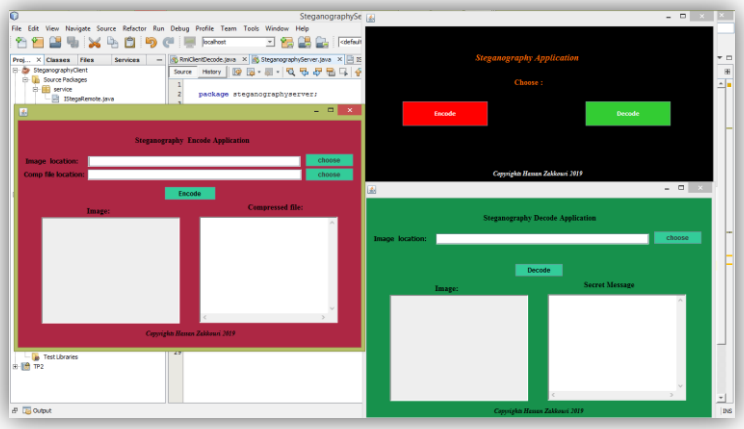
3) Interface :

Design :

En général, le design est pour les images qui peuvent être visualisé avec l'interface utilisateur java !

Ici, on a les images PGM qui nécessite l'utilisation d'un autre logiciel comme Matlab pour visualiser l'image.

En exécution : (Voir la vidéo pour la démarche d'utilisation)



4) Technique utilisé pour cacher les bits :

Premier problème : comment modifier les bits de poids faible ?

Pour passer de 00011110 à 00011100, il suffit de soustraire à 00011110 le reste de la division euclidienne de 00011110 par 2^n (ici n = 2 le nombre de bits par pixel), et pour passer de 00011100 à 00011101, il suffit d'ajouter 01.

```
86          // hiding two bits in a pixel
87          this.imgData[i][j] -=  this.imgData[i][j] % (Math.pow(2, n));
88          this.imgData[i][j] += twoBitsInt;
```

Récupération :

```
268             // retrieving two bits from a pixel
269             int temp = imgData[i][j];
270             pixels++;
271             temp = (int) (temp % Math.pow(2, n));
272             str[t++] = intToBin(temp, n);
```

Image originale :

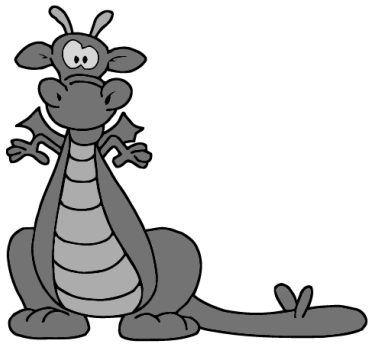
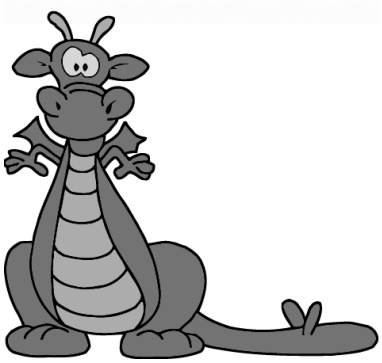


Image modifié :



La partie supérieure de l'image est sombré un peu comparant à l'image originale (n = 2).

IV. Conclusion :

Après avoir réalisé ce modeste travail, on a appris beaucoup de chose surtout dans les deux domaines, la stéganographie et la compression des données, à savoir :

- Comment compresser un texte avec l'algorithme LZW ;
- Comment manipuler les images au niveau de gris (PGM : Portable GrayMap) ;
- Comment utiliser la méthode LSB pour la stéganographie ;
- Comment implémenter une application client-serveur.

Comme perspective, nous envisageons de développer une autre application pour manipuler les images coloré RGB et d'améliorer l'interface !

</FIN>