Compte rendu TP TIM

Stéganographie et cryptographie

1. Lecture d'une image ppm

Le chargement d'une image s'effectue en 2 temps, la lecture et l'analyse de l'en tête ppm puis la lecture et le stockage des données (pixels) qui composent cette image.

- Lecture de l'entête :

L'objectif de cette étape est de récupérer le format de l'image (hauteur et largeur) présent dans l'entête. Le fichier est lu ligne par ligne, lorsqu'une ligne est détectée comme étant un commentaire (« # » en début de ligne) celle-ci est ignorée.

- Stockage des pixels :

La récupération de la taille de l'image dans l'entête nous permet d'allouer dynamiquement la mémoire pour le stockage des pixels. Ces pixels sont donc stockés au sein d'une matrice 2D dynamique. Pour chaque position (i, j) de la matrice on enregistre les valeurs RGB du pixel de même coordonnées dans l'image ppm. Le stockage des valeurs RGB s'effectue au sein d'une structure Pix_rgb, simplement composée de 3 unsigned chars (red, green et blue).
En somme notre image simplement stockée au sein d'une matrice de Pix_rgb.

Il nous est maintenant possible de modifier facilement chaque pixel de l'image en connaissant ses coordonnées. L'accès à ces pixels et d'autant plus simple que lors position dans la matrice correspond à leur position sur l'image ppm.

2. Dissimulation d'un message

Deux techniques de dissimulation ont été expérimentées.

2.1. Le premier procédé, en lien avec l'énoncé

Découpage de notre message, 2 bits par 2 bits, puis écriture de ces paires sur les deux bits de poids faibles d'un ensemble de pixels (bleus) de l'image.
Pour plus de discrétions, l'ensemble de pixels sur lesquelles est réparti le message n'est pas contigu et est évolutif. En effet, cet ensemble est déterminé par la fonction *Next_pixel()*. Cette fonction utilise les valeurs des pixels rouges et verts (+ cle vigenere si chiffrement) pour déterminer le nombre de pixels à passer avant le prochain stockage.

Next_pixel() :

A partir d'une position (i;j) donnée (ou *current_pos en 1D (0<*current_pos<img_size)), on calcule de combien de pixels nous allons avancer avant de cacher l'information suivante.

Calcul de l'avancement → *A = (red_pix(i;j) & green_pix(i:j)) / 5* *0 < A < 51*
current_pos += A + 1
si vigenere on ajoute → *current_pos += (current_pos & cleVig[i]) / 5*

Current_pos est donc strictement croissant. Les coordonnées (i;j) qui découlent de **tmp_pos** sont calculées en partant du début (**tmp_pos = current_pos**) ou de la fin de notre image (**tmp_pos = img_size – current_pos**). Les données sont donc cachées dans 2 zones distinctes (début et fin de l'image). L'établissement d'une taille maximale de message en fonction de l'image permet de garantir que ces 2 zones ne se chevauchent pas.

2.2. Le second procédé se base sur des automates cellulaire 2D

Cette partie s'inspire d'un papier issu l'université de Vidyasagar (merci à eux) : https://acadpubl.eu/jsi/2013-83-5/14/14.pdf

Pour rappel, un automate cellulaire 2D est un ensemble de cellules (des pixels par exemple) dont l'état à un instant T dépend de leur état propre et de celui de ses voisines à l'instant T-1.

Voisinage :
Le voisinage de nos cellules sera modélisé par le voisinage de Moore au rang r = 1. Concrètement, le voisinage d'une cellule sera limité aux huit cellules qui l'entourent

$$N^{M}_{(x_0,y_0)} = \{(x,y) : |x - x_0| \leq r, |y - y_0| \leq r\}$$

De ce voisinage nous pouvons établir une règle pour déterminer quelles cellules influeront dans le changement d'état de la cellule centrale.

32	64	128
16	0	1
8	4	2

Exemples :
- La règle 255 (1 + 2 + 4 + … + 128 **+ 0**) caractérise l'implication de toutes les cellules dans la détermination de l'état futur de la cellule centrale.

- La règle 131 (1 + 2 + 128 **+ 0**) caractérise l'implication des trois voisines de droites dans la détermination de l'état futur de la cellule centrale.

On note que la cellule 0 (la cellule centrale elle même) sera systématiquement impliquée dans le changement d'état ou non de cette même cellule, peu importe la règle choisie.

Dissimulation d'un message :
Le message à cacher est découpé en un ensemble de N bits. Les cellules sont représentées par le LSB de chaque pixel bleu de notre image. On parcours donc N cellules, et, on change l'état de chaque cellule (R) en fonction **de la parité de son voisinage** (voisins que l'on considère ou non en fonction de la règle établie) **et du bit** (cypher) **que l'on cherche à cacher** :

Cipher bit⇒ Parity⇓	$H = 0$	$H = 1$
Odd parity	Change R	No Change R
Even Parity	No Change R	Change R

Le décodage qui en résulte est simple, il suffit de parcourir notre matrice de pixels bleus, de calculer la parité du voisinage selon la règle établie, si le voisinage est pair alors le bit caché vaut 0, et respectivement 1 si le voisinage est impair.

Pour varier d'une image à l'autre, on se sert de la valeur du 1er pixel rouge pour déterminer la règle à appliquer (cf exemple ci contre):

	128	*64*	*32*	*16*	*8*	*4*	*2*	*1*	
first red value (8bit) →	*1*	*0*	*1*	*1*	*0*	*1*	*1*	*0*	*Pour cette image, on utilisera la règle 181*

Cette technique présente l'avantage d'être très discrète, le stockage de l'information n'implique pas nécessairement la modification du bit de l'image. De plus, le message ne peut pas être extrait par une simple lecture du LSB de chaque pixel puisqu'il dépend en vérité de la parité d'un ensemble de voisins.

3. Chiffrement Vigenère

Pour plus de sécurité encore, on implémente une fonction de chiffrement à notre outil stéganographique. Ce chiffrement et déchiffrement est indépendant des fonctions stéganographique, ce qui permet de le proposer en tant qu'option pour les deux modes de dissimulation (via automate cellulaire ou via LSB).

Les opérations de chiffrement ou de déchiffrement étant similaires, elle sont gérées au sein d'une unique fonction, prenant en paramètre l'action à effectuer (chiffrer ou déchiffrer).

README :

Compilation :
bash:~/tool/ make

Execution :
How to use this program :
Hide message into img :
./tp_tim -e -lsb <image.ppm>
-par <image.ppm>
You can crypt your message with -Vigenere at the end of the command

Recover message into img :
./tp_tim -d -lsb <image.ppm>
-par <image.ppm>
You must specify -Vigenere at the end of the command if a cypher text is hiding into the image

-par correspond à la dissimulation basée sur les automates cellulaire
-lsb correspond à la dissimulation sur les 2 bits de poids faible comme demandé dans l'énoncé

EXAMPLES :
./tp_tim -e -lsb chaton.ppm -Vigenere
./tp_tim -d -lsb chaton_enc.ppm -Vigenere