

Sujet:

Création d'un logiciel de stéganographie

Introduction:

La stéganographie est une technique qui permet de dissimuler les données secrètes dans un fichier à l'allure anodine. En d'autres termes, on peut insérer à l'intérieur d'une image ou d'un fichier son, ou texte, un fichier de son choix, quelle que soit sa nature.

Notre projet consiste à créer un logiciel de stéganographie qui permettra de dissimuler un message dans une image ou de détecter un message dans une image.

Sommaire:

- I. Présentation des objectifs et déroulement du codage
- II. Description des fonctions principales du programme
- III. Résultats obtenus
- IV. Conclusion
- V. Annexe : code

Organisation:

Yuhan DOU

Pauline LABOURIE

Jialin HAO

I. Présentation des objectifs et déroulement du codage

I.1 Objectifs

Le programme doit permettre d'effectuer les opérations suivantes :

- Estimation de la capacité de l'image porteuse.
- Dissimulation du message secret.
- Récupération du message secret.
- Optimisation de la capacité de l'image hôte.
- Détection automatique de la présence d'un message caché et sauvegarde du nom du fichier dissimulé.
- Sauvegarde du nom du fichier dissimulé.

I.2 Déroulement du codage

Nous disposons de deux fichiers : une image et un fichier texte. L'image servira de porteuse du message secret et le fichier texte contiendra le message secret.

Le programme commence par calculer la taille de l'image. Elle est fonction de sa capacité à dissimuler un message. Puis le programme demande à l'utilisateur quel est le nom du fichier texte contenant le message secret et calcule la taille du message, et la taille du nom du message. Si le message est trop grand par rapport à la taille de l'image il y aura une perte d'information.

Une structure encapsule des informations suivantes : la taille du message, la taille de l'image, le fichier contenant le message, le nom du fichier. Cette structure est ensuite convertie en une chaîne de caractères pour pouvoir être codée grâce à l'algorithme du LSB dans l'image.

La nouvelle image contenant le message secret est enfin créée.

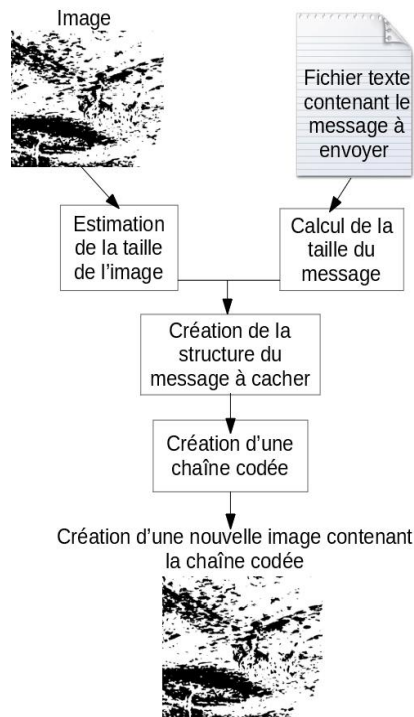


Schéma de principe du codage

I.3 Déroulement du décodage

Nous disposons d'une image étant potentiellement porteuse d'un message secret. Le programme commence par déterminer si l'image contient un message secret.

Grâce à l'algorithme du LSB, le programme récupère une chaîne de caractères. Cette chaîne est transposée dans une structure afin de structurer les données.

La structure encapsule des informations suivantes : la taille du message, la taille de l'image, le fichier contenant le message, le nom du fichier.

Ainsi nous pouvons retranscrire les informations du message secret décodé à l'utilisateur.

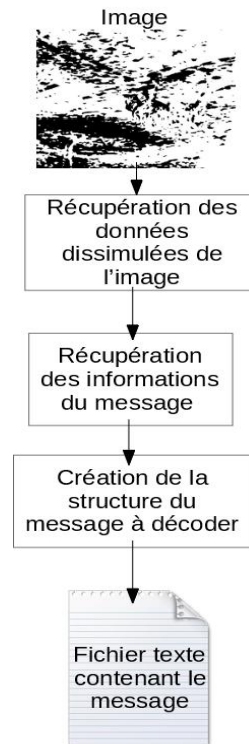


Schéma de principe du décodage

II. Description des fonctions principales du programme

II.1 EEALoadImage("lena.tif", image)

Fonction codée dans le programme d'origine fournit (tiftest) permettant d'importer une image.

II.2 int calcul_taille_image(bwimage_t *image)

Pour calculer la taille de l'image, on parcourt chaque pixel de l'image, et on incrémente la taille. La fonction prend en paramètre une image et retourne sa taille sous forme d'un entier.

II.3 int CRC(char *Adresse_tab, int Taille_max)

Permet de calculer le CRC du message. La fonction prend en paramètres une chaîne de caractères et la taille maximale du message. Elle retourne son CRC de type entier.

II.4 message init_message(int nb_char_msg, int taille_image, FILE * fichier, char nom [30])

Fonction d'initialisation d'une structure message. On calcule d'abord le nombre de caractères du fichier qui correspond à la taille des données. Puis on récupère le CRC, la taille des données, la taille de l'image, le nom du fichier et les données du fichier, on les encapsule dans une structure "message". La fonction retourne une nouvelle structure "message" initialisée.

typedef struct:

```
{  
  
unsigned int crc;  
  
unsigned char size_of_filename;  
  
unsigned int size_of_data;  
  
char *filename;  
  
char *data;  
  
}message;
```

II.5. void codage_image(message message_code, int taille_image, bwimage_t *image)

char *chaine_info(message info)

C'est la partie Codage:

II.5.1 En appelant la fonction "chaine_info", l'information de la structure de message "message_code" qui stocke le message secret (①CRC, ②Taille du nom du fichier, ③Taille de données du fichier, ④Nom du fichier, ⑤Données du fichier) est mise dans une chaîne (char)"ch ["]". Les éléments de "ch[]" stockent à leur tour CRC, Taille du nom du fichier, Taille du fichier, Nom du fichier et Données du fichier.

La taille de `ch []` peut être déduite de ①②③④⑤:

“①CRC”, “②Taille du nom du fichier”, “③Taille de données du fichier” occupent chacun un élément de `ch[]`, qui à son tour est `ch[0]`, `ch[1]` et `ch[2]`;

Le nombre total de caractères de “④Nom du fichier” est “②Taille du nom du fichier”, donc “④Nom du fichier” représentent `ch[3]` à `ch[3+Taille du nom du fichier-1]`;

Le nombre total de caractères de “⑤Données du fichier” est “③Taille de données du fichier”, donc “⑤Données du fichier” représentent `ch[3+Taille du nom du fichier]` à `ch[3+Taille du nom du fichier+Taille de données du fichier-1]`;

Donc, la taille de `ch[]` est égale à “3+Taille du nom du fichier+Taille de données du fichier”.

II.5.2 L’algorithme LSB consiste à coder l’information à cacher dans les bits les moins significatifs de l’intensité (Least Significant Bits, d’où le nom). Ici, l’on utilise la valeur du dernier bit pour le codage. Puisque les éléments de `ch[]` sont des caractères du type “char”, et chaque caractère de l’ordinateur est constitué d’un code ascii binaire de 8 bits. Dans ce cas, pour les images en noir et blanc à 256 niveaux de gris, chaque pixel est composé de 8 bits. Donc, la capacité de dissimulation est égale à 1/8 de la taille de l’image. Chaque caractère est divisé en 8 chiffres binaires indépendants (0 ou 1) ajoutés à 8 pixels.

000 000 000	000 000 001
001 000 001	111 111 111

Avant codage

001 001 001	001 001 001
001 000 001	111 111 111

Après codage

Afin de remplacer le dernier bit d’un pixel par du code, une "opération d’effacement" est nécessaire à la fin du pixel. C’est-à-dire qu’on effectue “AND” de chaque pixel (`image->data[i][j]`) et 1111 1110 (0xfe).

II.5.3 Avec l’opération "séparation par bit", on divise chaque caractère en 8 chiffres binaires séparés (0 ou 1). Pour chaque caractère du message (chaque élément de `ch[]`), on fait ce qui suit:

(1) $Ch[m]$ décale à gauche de n bits.

m ($0 \leq m < \text{taille de } ch[]$) représente le $(m+1)^{\text{ème}}$ élément de $ch[]$, qui est le $(m+1)^{\text{ème}}$ caractère du message; n ($0 \leq n < 8$) représente le $n^{\text{ème}}$ chiffre binaire lors de la représentation d'un caractère avec un code ascii.

(2) Ensuite, on effectue "AND" sur le résultat de l'étape (1) avec 1000 0000 (0x80), et on retire le premier bit. C'est le $n^{\text{ème}}$ chiffre binaire du code ascii.

(3) Ensuite, le résultat de l'étape (2) décale à droite de 7 bits. Après, on l'effectue "AND" avec 0000 0001 (0x01) pour déplacer chaque chiffre binaire du code ascii vers le dernier bit.

En fait, on effectue "AND" avec 0000 0001 (0x01) afin de s'assurer que les 7 premiers bits sont tous effacés, parce que la décalage à droite est équivalente à la expansion du premier bit. Par exemple, après que "1000 0000" décale à droite de 7 bits, on obtient "1111 1111", mais pas "0000 0001".

(4) Chaque fois que le résultat de l'opération ci-dessus est stocké dans un autre tableau $temp[]$, donc chaque élément de $temp[]$ est le chiffre binaire du code ascii correspondant à chaque caractère dans $ch[]$. L'opération "séparation par bit" est finie.

II.5.4 En fin, on additionne chaque pixel après "l'opération d'effacement" et chaque élément du $temp[]$ qui est effectué l'opération "séparation par bit". De cette façon, nous avons accompli le codage.

II.5.5 Puisque le codage peut ne pas nécessiter tous les pixels, avant l'opération d'effacement, il faut calculer quels pixels doivent être effacés en fonction de la longueur de $ch[]$.

On pose que le codage suit la règle "Priorité de ligne". On définit les deux constantes suivantes:

$$I = (\text{taille_message} * 8) / \text{image} \rightarrow \text{width};$$

$$J = (\text{taille_message} * 8) \% \text{image} \rightarrow \text{width};$$

Le codage est divisé en deux parties:

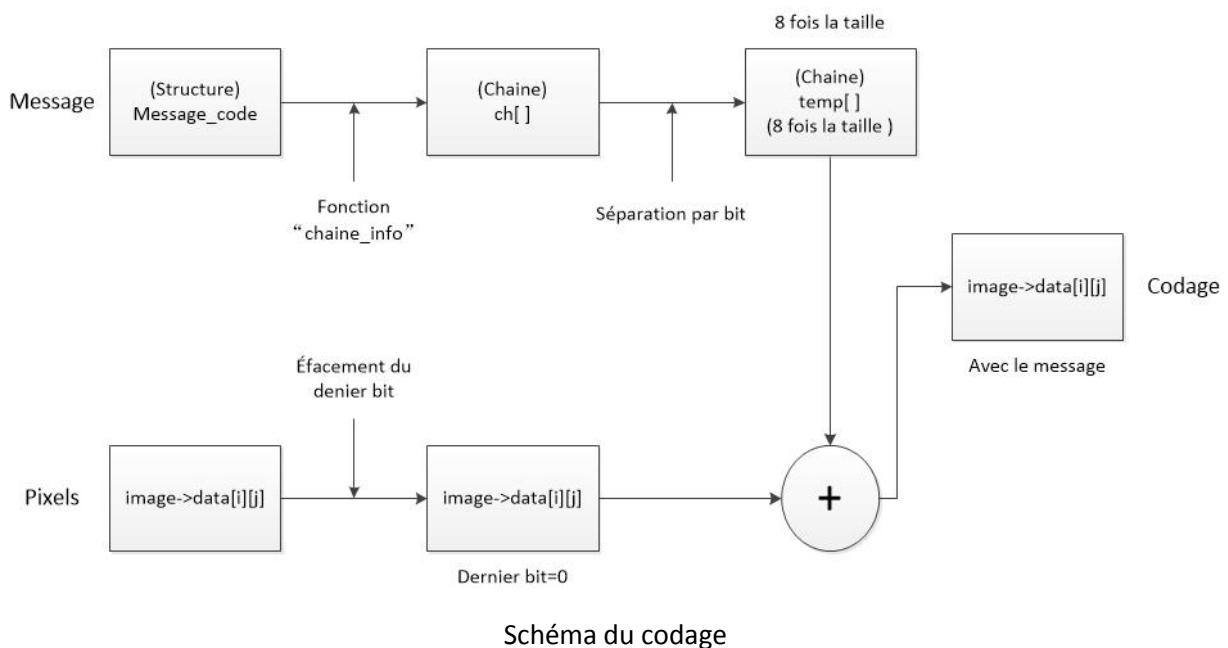
(1) $0 \leq i < I$, $0 \leq j < \text{image} \rightarrow \text{width}$

(2) $i = I$, $0 \leq j < J$

Pour chaque pixel $\text{image} \rightarrow \text{data}[i][j]$, le code qui lui est ajouté correspond à $\text{temp}[k]$ $= \text{temp}[(\text{image} \rightarrow \text{width}) * i + j]$.

i,j	0,0	0,1	...	0,(image->width)-1
k	0=(image->width)*0+0	0=(image->width)*0+1		0=(image->width)*0+(image->width)-1
i,j	1,0	1,1	...	1,(image->width)-1
k	(image->width)*1+0	(image->width)*1+1		(image->width)*1+(image->width)-1

La fonction `codage_image()` prend une structure de type "message" et la taille de l'image. Elle crée un nouveau image avec un message secret dedans.



II.6 char *decodage_image(int taille_image, bwimage_t *image)

II.6.1 On définit un tableau (char) "chainedecode[]" pour stocker chaque caractère du message décodé. Afin de déterminer la taille de chainedecode[], on decode d'abord le "① CRC", "② Taille du nom du fichier" et "③ Taille de données du fichier". Alors la taille de chainedecode[] est égale à "3+Taille du nom du fichier+Taille de données du fichier".

II.6.2 (1) On effectue "AND" de chaque pixel (image->data[i][j]) avec 0000 0001 (0x01) pour retirer son dernier bit. Les résultats sont stockés dans un tableau à 2 dimension "newdata[i][j]".

D'après l'algorithme de codage, nous pouvons voir que chaque 8 éléments de newdata[i][j] peut être combiné en un caractère du message, qui est un élément de chainedecode[]. On définit un variable "k" pour indiquer le k^{ème} élément dans le codechaîne[].

(2) Chaque 8 colonnes étant un groupe, chaque ligne est divisée en “nb_cara_par_ligne = (image->width)/8” groupes, et chaque groupe correspond à un caractère.

On définit deux variables m et n. m indique le m^{ème} groupe de chaque ligne, et n représente le n^{ème} élément de chaque groupe.

(3) Le n^{ème} élément de chaque groupe décale à gauche de 7-n bits.

(4) On effectue “OR” du résultat du décalage de tous les 8 éléments de chaque groupe, ça donne un code ascii d’un caractère du message.

II.6.3 Selon la taille de codechaîne[] (3+Taille du nom du fichier+Taille de données du fichier), le nombre de pixels à décoder peut être calculé.

Le codage suit la règle "Priorité de ligne", donc le décodage suivi aussi cette règle. Suppose que le nombre des groupes par ligne est nb_cara_par_ligne, on définit les deux constantes suivantes:

$$l = (3 + \text{taille_nom} + \text{taille_data}) / \text{nb_cara_par_ligne};$$

$$M = (3 + \text{taille_nom} + \text{taille_data}) \% \text{nb_cara_par_ligne};$$

Le décodage est divisé en deux parties:

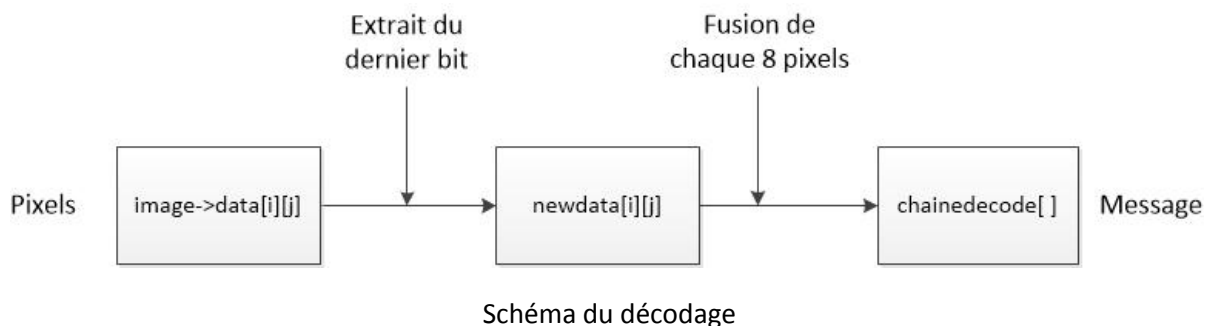
(1) $0 \leq i < l$, $0 \leq m < \text{nb_cara_par_ligne}$

(2) $i = l$, $0 \leq m < M$

Pour chaque pixel image->data[i][j], la relation entre l’indice de la colonne (j), du groupe (m) et du élément dans chaque groupe (n) est: $j = 8 * m + n$

Donc, la relation entre le pixel et chainedecode[k] est: $k = \text{nb_cara_par_ligne} * i + m$

i=0	m=0							
	n=0	n=1	n=2	n=3	n=4	n=5	n=6	n=7
	k=0=nb_cara_par_ligne*0+0							
i=0	m=1							
	n=0	n=1	n=2	n=3	n=4	n=5	n=6	n=7
	k=1=nb_cara_par_ligne*0+1							
... ..								
i=1	m=0							
	n=0	n=1	n=2	n=3	n=4	n=5	n=6	n=7
	k=nb_cara_par_ligne*1+0							
i=1	m=1							
	n=0	n=1	n=2	n=3	n=4	n=5	n=6	n=7
	k=nb_cara_par_ligne*1+1							



II.7 void detection_message(char *pch)

Après décodage, on obtient une chaîne de caractères. Pour détecter s'il y a un message, on prend le quatrième élément de la chaîne, avec ceux qui suivent jusqu'à "3+taille_nom". Ils composent le nom du fichier origine. Puis on les compare avec les noms des tous les fichiers qu'on a. On introduit un flag pour marquer le résultat de comparaison. S'il y a un fichier correspondant, on affiche le nom du fichier et on prend successivement les éléments

suivants qui sont des données du fichier. Sinon, on informe l'utilisateur que l'image ne contient pas de message secret.

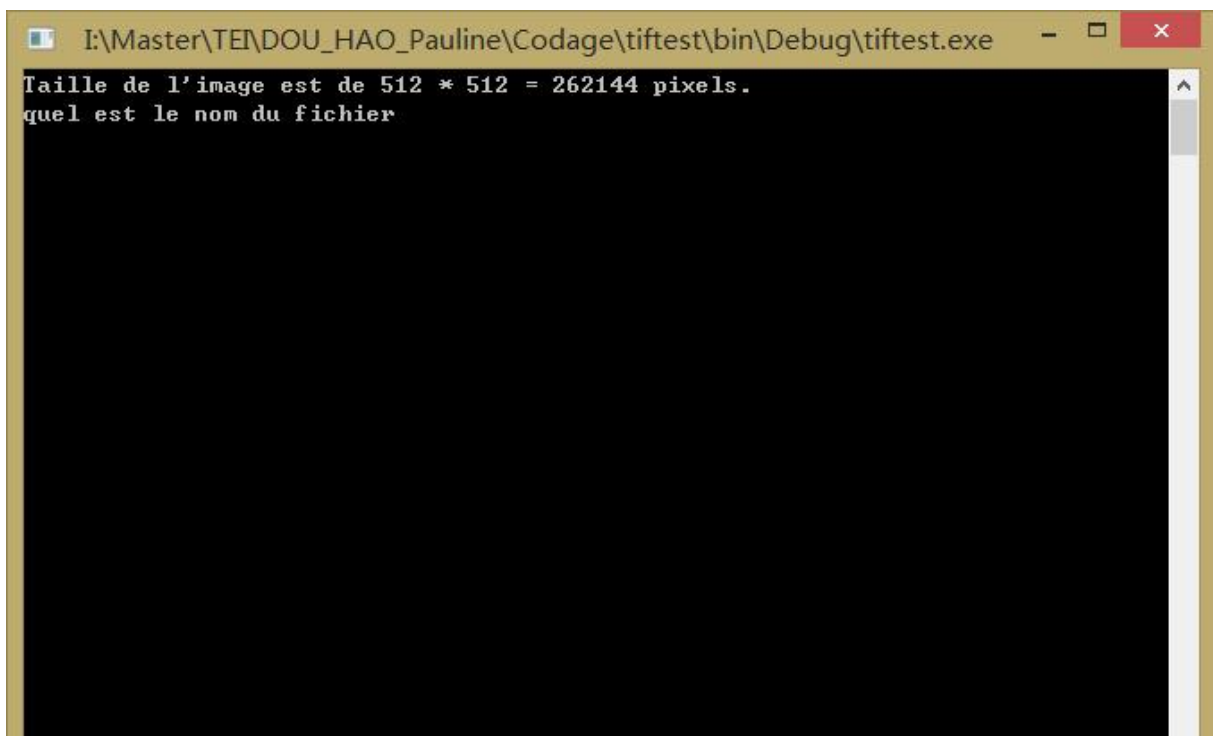
II.8 void sauvegarde(char *message)

La fonction sauvegarde() prend en paramètres une chaîne de caractères. Elle demande à l'utilisateur le nom sous lequel on sauvegarde le fichier. Puis elle crée un fichier, met les données dans le fichier et renomme le fichier.

III. Résultats obtenus

III.1 Codage

Après le codage, on obtient le résultat suivant:



La taille de l'image est $512 * 512 = 262144$ pixels.

Ensuite, on entre le nom du fichier contenant le message que on souhaite ajouter dans l'image, et on obtient le résultat suivant:

```

I:\Master\TEI\DOU_HAO_Pauline\Codage\tifttest\bin\Debug\tifttest.exe
Taille de l'image est de 512 * 512 = 262144 pixels.
quel est le nom du fichier
fichier_secret_3

CRC:
2d

Taille du nom:
20

Taille de donnees:
57

Nom du fichier:
fichier_secret_3.txt

Donnees du fichier:
I opened my eyes last night and saw you in the low light.

Process returned 0 (0x0) execution time : 9.628 s
Press any key to continue.

```

On peut obtenir toutes les informations du fichier : le CRC, la taille du nom, la taille des données, le nom du fichier du message et les données du fichier.

On ouvre le dossier où se trouve le programme du codage, et observe qu'il y a une nouvelle image nommée "imagecode(LSB).tif".

这台电脑 > DONNA (I:) > Master > TEI > DOU_HAO_Pauline > Codage > tifttest				
名称	修改日期	类型	大小	
obj	2018/3/30 11:53	文件夹		
bin	2018/3/30 11:53	文件夹		
libjpeg.dll	2005/5/15 15:08	应用程序扩展	125 KB	
lena	2000/6/12 11:33	看图王 TIF 图片文...	258 KB	
imagecode(LSB)	2018/4/1 22:37	看图王 TIF 图片文...	258 KB	



l'image originale (lena.tif)

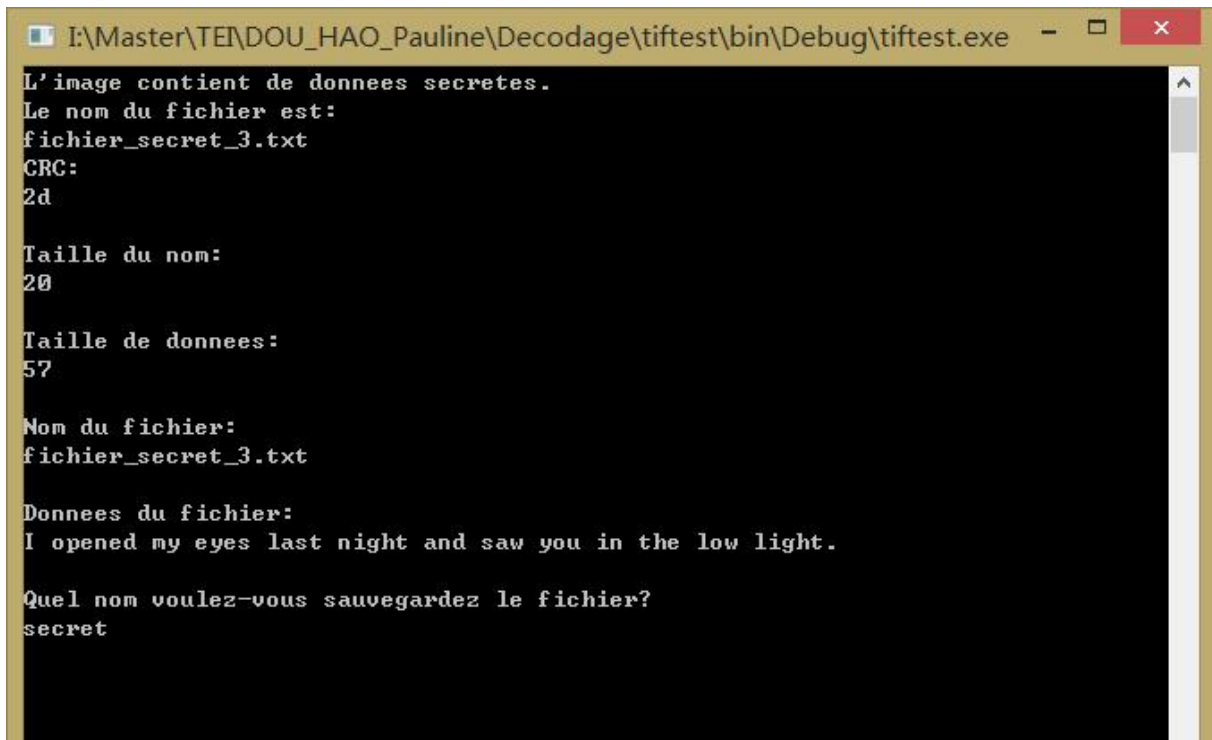


l'image avec le message (imagecode(LSB.tif))

On peut à peine distinguer la différence entre les deux images. Donc, le programme du codage fonctionne bien!

III.2 Détection et décodage

On fait le décodage sur l'image "imagecode(LSB).tif", et obtient le résultat suivant:



```

I:\Master\TEI\DOU_HAO_Pauline\Decodage\tiftest\bin\Debug\tiftest.exe
L'image contient de donnees secretes.
Le nom du fichier est:
fichier_secret_3.txt
CRC:
2d

Taille du nom:
20

Taille de donnees:
57

Nom du fichier:
fichier_secret_3.txt

Donnees du fichier:
I opened my eyes last night and saw you in the low light.

Quel nom voulez-vous sauvegardez le fichier?
secret

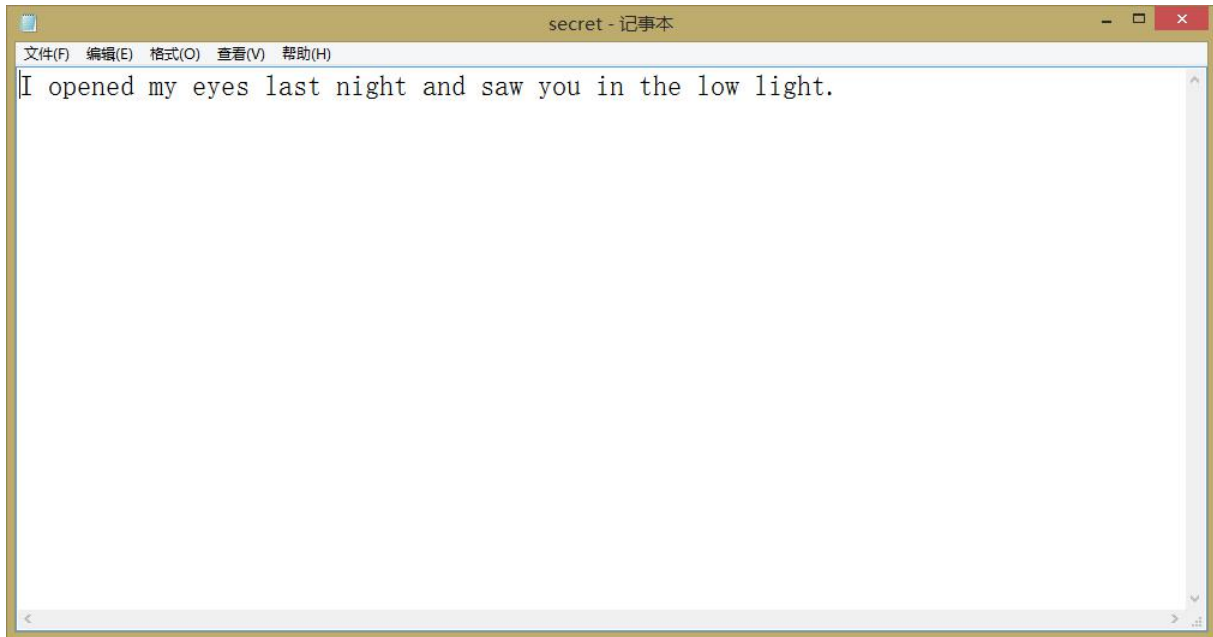
```

Le programme informe l'utilisateur que le fichier "imagecode(LSB).tif" contient un fichier dissimulé. Il contient aussi toutes les informations du message dans l'image: CRC, la taille du nom du fichier, la taille de données du fichier, le nom du fichier et les données du fichier. Par comparaison aux informations précédentes dans le codage, on voit que les informations du message sont correctes. Ensuite, le programme demande à l'utilisateur le nom sous lequel il souhaite sauvegarder le fichier, et on entre "secret".

On ouvre le dossier où se trouve le programme du décodage, et on observe qu'il y a un nouveau fichier "secret.txt".

这台电脑 > DONNA (I:) > Master > TEI > DOU_HAO_Pauline > Decodage > tiftest			
名称	修改日期	类型	大小
obj	2018/3/30 11:53	文件夹	
bin	2018/3/30 11:53	文件夹	
libjpeg.dll	2005/5/15 15:08	应用程序扩展	125 KB
imagecode(LSB)	2018/4/1 22:37	看图王 TIF 图片文...	258 KB
secret	2018/4/1 22:40	TXT 文件	1 KB

Il contient le message suivant:



C'est bien le message que l'on souhaite ajouter dans l'image "lena.tif". Le programme de détection et du décodage fonctionne bien!

IV. Conclusion

Lors de ce projet en informatique, nous avons pu découvrir le principe de fonctionnement de la stéganographie. Cette technique de dissimulation de message engendre de nombreuses connaissances sur la manipulation des fichiers.

Nous avons pu approfondir nos connaissances dans la manipulation des fonctions et des structures dans le langage C. Et élargir nos compétences dans les techniques propres à la stéganographie comme dans les codages de pixels ou les algorithmes de manipulation des bits.

Dans notre projet, nous avons uniquement dissimulé des informations de fichiers textes dans des images, mais la stéganographie peut s'étendre aussi bien à d'autres types de fichiers tels que les fichiers audios ou vidéos.

V. Annexe : le code du programme

V.1 main (pour codage)

```
#include "../tifwrap/tifwrap.h"
#include "message.h"
int CRC(char *ptr, int taille_data);
char *chaine_info(message info);
message init_message(int nb_char_msg, int taille_image, FILE * fichier, char nom [30]);
void codage_image(message message_code, int taille_image, bwimage_t *image);
int calcul_taille_image(bwimage_t *image);
int main()
{
    bwimage_t *image;
    error_e retval=EEA_OK;
    image=EEACreateImage();
    do
    {
        // Recherche de l'image
        if(EEA_OK != (retval=EEALoadImage("lena.tif", image))) break;
        // Calcul taille de l'image
        int taille_image;
        taille_image = calcul_taille_image(image);
        printf("Taille de l'image est de %d * %d = %dpixels.\n",image->height,image->width,taille_image);
        //Recherche du fichier secret
        char nom[30];
        printf("quel est le nom du fichier\n");
        scanf("%s",nom);
        FILE *fichier;
        char str[5]=".txt";
        strcat(nom,str);
        fichier=fopen(nom,"r");
        // Calcul de la taille du message
        int nb_char_msg=0;
        while(fgetc(fichier)!=EOF)nb_char_msg++;
        fclose(fichier);
        // Création de la structure message
        message message_code = init_message(nb_char_msg, taille_image, fichier, nom);
        // CODAGE DE L'IMAGE
```



```
    codage_image( message_code,  taille_image, image);
    // Création de la nouvelle image codée
    if(EEA_OK != (retval=EEADumpImage("imagecode(LSB).tif", image))) break;
}
while(0);
switch(retval)
{
case EEA_OK:
    break;
case EEA_ENOFILE:
    fprintf(stderr, "Cannot open file\n");
    break;
case EEA_EBADBPS:
    fprintf(stderr, "Number of bits per sample must be equal to 8\n");
    break;
case EEA_EBADPHOTOMETRIC:
    fprintf(stderr, "Not a colormap image\n");
    break;
case EEA_ENOCOLORMAP:
    fprintf(stderr, "Image does not have a colormap\n");
    break;
case EEA_ENOTGRAY:
    fprintf(stderr, "At least one entry in the colormap is not gray\n");
case EEA_ENOMEM:
    fprintf(stderr, "Cannot allocate memory\n");
    break;
default:
    ;/* Can't happen */
}
EEAFreeImage(image);
return 0;
}
```

V.2 taille_image.c

```
#include "message.h"
int calcul_taille_image(bwimage_t *image)
{
    int i,j;
```

```
    int taille_image=0;
    // Calcul de la capacité de l'image
    for(i=0; i<image->height; i++)
    {
        for(j=0; j<image->width; j++)
        {
            taille_image++;
        }
    }
    return taille_image;
}
```

V.3 message.h

```
#ifndef MESSAGE_H_INCLUDED
#define MESSAGE_H_INCLUDED
#include <stdio.h>
// Déclaration de la structure message
typedef struct{
    unsigned int crc;
    unsigned int size_of_filename;
    unsigned int size_of_data;
    char *filename;
    char *data;
}message;
```

V.4 message.c

```
#include "message.h"
#include "../tifwrap/tifwrap.h"
int CRC(char *ptr, int taille_data);
message init_message(int nb_char_msg, int taille_image, FILE * fichier, char nom [30])
{
    message message_code;
    char data[taille_image];

    int size_struct_msg = sizeof(message_code.crc) + sizeof(message_code.size_of_filename) +
    sizeof(message_code.size_of_data) + sizeof(message_code.filename) + sizeof(message_code.data);

    // Vérification de la taille du message à envoyer
    if ( nb_char_msg>(taille_image-size_struct_msg) )
    {
```

```
    printf("Message secret trop long\n");
}
// Récupération de la chaîne de caractère du message secret à envoyer
else
{
    fichier=fopen(nom,"r");
    int i=0;
    char caractereActuel;
    // Boucle de lecture des caractères un à un
    do
    {
        caractereActuel = fgetc(fichier);
        data[i]=caractereActuel;
        i++;
    } // On continue tant que fgetc n'a pas retourné EOF (fin de fichier)
    while (caractereActuel != EOF);
    // Affichage du message secret
    /*for(i=0; i<nb_char_msg; i++)
    {
        printf("%c", data[i]);
    }*/
}
// Mettre sous forme de structure le message secret du fichier
message_code.crc = CRC(data,nb_char_msg);
message_code.size_of_filename = strlen(nom);
message_code.size_of_data = nb_char_msg;
message_code.filename = nom;
message_code.data = data;
return message_code;
}
```

V.5 CRC.c

```
#include "message.h"
#include "../tifwrap/tifwrap.h"
int CRC(char *ptr, int taille_data)
{
    int i;
    int crc=0;    while(taille_data--)
```

```
{
    crc ^= *ptr++;
    for (i=8; i>0; --i)
    {
        if (crc & 0x80)
            crc = (crc << 1) ^ 0x31;
        else
            crc = (crc << 1);
    }
}
return (crc);
}
```

V.6 chaine_info.c

```
#include "message.h"
#include "../tifwrap/tifwrap.h"
char *chaine_info(message info)
{
    char infotot[info.size_of_filename+info.size_of_data+3];
    infotot[0]= info.crc;
    infotot[1]= info.size_of_filename;
    infotot[2]= info.size_of_data;
    int i;
    for(i=3; i<3+info.size_of_filename; i++)
    {
        infotot[i]=*info.filename++;
    }
    for(i=3+info.size_of_filename; i<info.size_of_filename+info.size_of_data+3; i++)
    {
        infotot[i]=*info.data++;
    }
    printf("\nCRC:\n%x\n\nTaille du nom:\n%d\n\nTaille de donnees: \n%d\n",infotot[0],infotot[1],
infotot[2]);
    printf("\nNom du fichier:\n");
    for(i=3; i<(3+info.size_of_filename); i++)
    {
        printf("%c",infotot[i]);
    }
}
```

```
printf("\n\nDonnees du fichier:\n");
for(i=3+info.size_of_filename; i<(info.size_of_filename+info.size_of_data+3); i++)
{
    printf("%c",infotot[i]);
}
printf("\n");
char *infos=infotot;
return infos;
}
```

V.7 codage.c

```
#include "message.h"
#include "../tifwrap/tifwrap.h"
char *chaine_info(message info);
void codage_image(message message_code, int taille_image, bwimage_t *image)
{
    // Mettre la structure message sous forme de chaine de caractère
    unsigned int i,j;
    int taille_message;
    taille_message = 3 + message_code.size_of_filename + message_code.size_of_data;
    char ch[taille_message],*pch;
    pch=chaine_info(message_code);
    int g=0;
    for(g=0; *pch!='\0'; g++)
    {
        ch[g]=*pch;
        pch++;
    }

    // CODAGE DE L'IMAGE
    int m,n,l,J;
    int temp[8*taille_message];
    for(m=0; m<taille_message; m++)
    {
        for(n=0; n<8; n++)
            temp[8*m+n]=((0x80&(ch[m]<<n))>>7)&0x01;
    }
    l=(taille_message*8)/image->width;
```

```
J=(taille_message*8)%image->width;
for(i=0; i<I; i++)
{
    for(j=0; j<image->width; j++)
    {
        image->data[i][j]=0xfe&image->data[i][j];
        image->data[i][j]=image->data[i][j]+temp[(image->width)*i+j];
    }
}
for(j=0; j<J; j++)
{
    image->data[I][j]=0xfe&image->data[i][j];
    image->data[I][j]=image->data[I][j]+temp[(image->width)*I+j];
}
}
```

V.8 main (pour decodage)

```
#include "../tifwrap/tifwrap.h"
int calcul_taille_image(bwimage_t *image);
char *decodage_image(int taille_image, bwimage_t *image);
int detection(char *chainedecode, int taille_nom);
void sauvegarde(char *message);
int main()
{
    bwimage_t *image;
    error_e retval=EEA_OK;
    image=EEACreateImage();
    do
    {
        // Recherche de l'image
        if(EEA_OK != (retval=EEALoadImage("imagecode(LSB).tif", image))) break;

        // Calcul taille de l'image
        int taille_image;
        taille_image = calcul_taille_image(image);

        //decodage
        char *pch,chainedecode[taille_image];
```

```
pch=decodage_image(taille_image, image);
int g=0;
for(g=0; *pch!='\0'; g++)
{
    chainedecode[g]=*pch;
    pch++;
}
int taille_nom,taille_data;
taille_nom=(int)chainedecode[1];
taille_data=(int)chainedecode[2];

int count;
count=detection(chainedecode, taille_nom);

//Affichier les details du message secret
if(count!=0)
{
    printf("\nCRC:\n%x\n\nTaille du nom:\n%d\n\nTaille de donnees:\n%d\n\n",
chainedecode[0],chainedecode[1],chainedecode[2]);
    printf("Nom du fichier:\n");
    int i;
    for(i=3;i<3+taille_nom;i++)
    {
        printf("%c",chainedecode[i]);
    }
    printf("\n\nDonnees du fichier:\n");
    for(i=3+taille_nom;i<3+taille_nom+taille_data;i++)
    {
        printf("%c",chainedecode[i]);
    }
    printf("\n\n");

    //demander a l'utiliser le nom du fichier
    char message_data[taille_data];
    for(i=0;i<taille_data;i++)
    {
        message_data[i]=chainedecode[i+3+taille_nom];
    }
}
```

```
        sauvegarde(message_data);
    }
}
while(0);
switch(retval)
{
case EEA_OK:
    break;
case EEA_ENOFILE:
    fprintf(stderr, "Cannot open file\n");
    break;
case EEA_EBADBPS:
    fprintf(stderr, "Number of bits per sample must be equal to 8\n");
    break;
case EEA_EBADPHOTOMETRIC:
    fprintf(stderr, "Not a colormap image\n");
    break;
case EEA_ENOCOLORMAP:
    fprintf(stderr, "Image does not have a colormap\n");
    break;
case EEA_ENOTGRAY:
    fprintf(stderr, "At least one entry in the colormap is not gray\n");
case EEA_ENOMEM:
    fprintf(stderr, "Cannot allocate memory\n");
    break;
default:
    /* Can't happen */
}
EEAFreeImage(image);
return 0;
}
```

V.9 decodage.c

```
#include "../tifwrap/tifwrap.h"
char *decodage_image(int taille_image, bwimage_t *image)
{
    int i,j;
    int newdata[image->height][image->width];
```



```
char chainedecode[taille_image/8];
for(i=0; i<image->height; i++)
{
    for(j=0; j<image->width; j++)
    {
        newdata[i][j]=0x01&image->data[i][j];
    }
}
//decodage sur CRC, taille du nom, taille de données
int m,n;
for(m=0;m<3;m++)
{
    chainedecode[m]=0;
    for(n=0;n<8;n++)
    {
        newdata[0][8*m+n]=newdata[0][8*m+n]<<(7-n);
        chainedecode[m]=chainedecode[m] | newdata[0][8*m+n];
    }
}
//decodage sur filename et data
int taille_nom,taille_data;
taille_nom=(int)chainedecode[1];
taille_data=(int)chainedecode[2];
char subchaine[3+taille_nom+taille_data];
int nb_cara_par_ligne=(image->width)/8;
int l,M;
l=(3+taille_nom+taille_data)/nb_cara_par_ligne;
M=(3+taille_nom+taille_data)%nb_cara_par_ligne;
//decodage sur l'autre
for(i=0;i<l;i++)
{
    for(m=0;m<nb_cara_par_ligne;m++)
    {
        subchaine[nb_cara_par_ligne*i+m]=0;
        for(n=0;n<8;n++)
        {
            newdata[i][8*m+n]=newdata[i][8*m+n]<<(7-n);

subchaine[nb_cara_par_ligne*i+m]=subchaine[nb_cara_par_ligne*i+m] | newdata[i][8*m+n];
```

```
        }
    }
}
for(m=0;m<M;m++)
{
    subchaine[nb_cara_par_ligne*l+m]=0;
    for(n=0;n<8;n++)
    {
        newdata[l][8*m+n]=newdata[l][8*m+n]<<(7-n);

        subchaine[nb_cara_par_ligne*l+m]=subchaine[nb_cara_par_ligne*l+m]|newdata[l][8*m+n];
    }
}
for(i=3;i<3+taille_nom+taille_data;i++)
{
    chainedecode[i]=subchaine[i];
}
char *p=chainedecode;
return p;
}
```

V.10 detection.c

```
#include "../tifwrap/tifwrap.h"
#include <string.h>
int detection(char *chainedecode, int taille_nom)
{
    char *nom_message1="fichier_secret_1.txt";
    char *nom_message2="fichier_secret_2.txt";
    char *nom_message3="fichier_secret_3.txt";
    char nom_message[taille_nom];
    int i;
    for(i=0;i<taille_nom;i++)
    {
        nom_message[i]=chainedecode[i+3];
    }
    nom_message[taille_nom]='\0';
    int flag[3],count;
    flag[0]=strcmp(nom_message1,nom_message);
```

```
flag[1]=strcmp(nom_message2,nom_message);
flag[2]=strcmp(nom_message3,nom_message);

if(flag[0]==0)
    count=1;
else if(flag[1]==0)
    count=2;
else if(flag[2]==0)
    count=3;
else
    count=0;
switch(count)
{
    case 0:
        printf("L'image ne contient aucune donnee secrete.\n");
        break;
    case 1:
        printf("L'image contient de donnees secretees.\nLe nom du fichier est:\n");
        for(i=0;i<20;i++)
        {
            printf("%c",nom_message1[i]);
        }
        break;
    case 2:
        printf("L'image contient de donnees secretees.\nLe nom du fichier est:\n");
        for(i=0;i<20;i++)
        {
            printf("%c",nom_message2[i]);
        }
        break;
    case 3:
        printf("L'image contient de donnees secretees.\nLe nom du fichier est:\n");
        for(i=0;i<20;i++)
        {
            printf("%c",nom_message3[i]);
        }
        break;
    default:
```

```
        printf("Error!\n");
        break;
    }
    return count;
}
```

V.11 sauvegarde.c

```
#include "../tifwrap/tifwrap.h"
#include "message.h"
void sauvegarde(char *message)
{
    char nom[100];
    char hz[]=".txt";
    printf("Quel nom voulez-vous sauvegardez le fichier?\n");
    gets(nom);
    strcat(nom,hz);
    FILE *fp ;
    fp=fopen(nom,"w+");
    fprintf(fp,message);
    fclose(fp);
}
```