**Codage:**

1. 通过调用“chaine\_info”函数，将存放message secret的message结构体“message\_code”中关于message的信息（①CRC，②Taille du nom du fichier，③Taille de données du fichier，④Nom du fichier，⑤Données du fichier）放入一个char型数组“ch[ ]”。ch[ ]的元素依次存放CRC、Taille du nom du fichier、Taille de données du fichier、Nom du fichier、Données du fichier。

En appelant la fonction "chaine\_info", l'information de la structure de message "message\_code" qui stocke le message sécret (①CRC，②Taille du nom du fichier，③Taille de données du fichier，④Nom du fichier，⑤Données du fichier) est mise dans un tableau(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.

ch[ ]的长度可由信息①②③④⑤得出：

1. ①CRC，②Taille du nom du fichier，③Taille de données du fichier各占ch[ ]的一个元素，依次为ch[0]、ch[1]、ch[2]；
2. ④Nom du fichier的字符总数为②Taille du nom du fichier，因此占ch[3]到ch[3+Taille du nom du fichier-1]；
3. ⑤Données du fichier的字符总数为③Taille de données du fichier，因此占ch[3+Taille du nom du fichier]到ch[3+Taille du nom du fichier+Taille de données du fichier-1]；
4. 因此，ch[ ]数组的长度为3+Taille du nom du fichier+Taille de données du fichier。

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

1. “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];
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];
3. 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];
4. Donc, la taille de ch[] est égale à “3+Taille du nom du fichier+Taille de données du fichier”.
5. LSB的算法是将message加在图像每一个像素点的最不重要的一位，这里，我们加在最后一位。由于ch[ ]的元素为char型字符，在计算机中每个字符由8位二进制的ascii码组成，且256级灰度图像的每个像素也由8个bit组成，因此message的最大长度等于像素个数的1/8。每一个字符被拆分成独立的8个二进制数（0或1）添加在8个像素点上。

为了将像素的最后一个bit用code代替，需要对像素的末尾进行“**清零操作**”。即将每个像素image->data[i][j]和0000 0001（0x01）进行与运算。

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 “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.

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]) avec 0000 0001 (0x01).

1. 通过“**按位分离**”操作，将每一个字符拆分成独立的8个二进制数（0或1）。

对于message的每一个字符，即ch[ ]的每一个元素，进行如下操作：

1. ch[m]左移n位。

m（0≤m<taille de ch[ ]）表示ch[ ]的第m+1个元素，即message的第m+1个字符；n(0≤n<8)表示用ascii码表示一个字符时，第n位二进制数。

1. 接着，和1000 0000（0x80）进行与运算，取出最高位，即ascii码的各位二进制数。
2. 右移7位，并和0000 0001（0x01）进行与运算，将ascii码的每位二进制数移至最低位。

和0x01进行与运算是为了保证前7位都清零。右移是进行最高位的扩充，例如：

1000 0000右移7位之后的结果为1111 1111，而非0000 0001。

1. 将每次以上操作的结果存入另一数组temp[ ]中，则temp[ ]中的每一个元素是ch[ ]中各字符对应的ascii码的各位数值，“按位分离”操作完成。

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

1. Ensuite, on effecte “AND” sur le résultat de l’étape (1) avec 1000 0000 (0x80), et on retire le premier bit. C’est le nième chiffre binaire du code ascii.
2. Ensuite, le résultat de l’étape (2) décale à droite de 7 bits. Après, on l’effecte “AND” avec 0000 0001 (0x01) pour déplacer chaque chiffre binaire du code ascii vers le dernier bit .

En fait, on effecte “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 expasion du premier bit. Par exemple, après que “1000 0000” décale à droite de 7 bits, on obtient “1111 1111”, mais pas “0000 0001”.

1. 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.

例：

m=0，ch[0]=’A’，其ascii码为0100 0001。

n=0: ①右移n=0位:

0100 0001 >> 0 = **0**100 0001

②和0x80与运算：

**0**100 0001 & 1000 0000 = **0**000 0000

③左移7位：

**0**000 0000 << 7 = 0000 000**0**

④和0x01与运算：

0000 000**0** & 0000 0001 = 0000 000**0**

此时，取出ch[0]的ascii码的第1位。因此，temp[8\*m+n]=temp[0]的ascii码为0000 0000，

即temp[0]=0。

n=1: ①右移n=1位:

0100 0001 >> 1 = **1**000 0010

②和0x80与运算：

**1**000 0010 & 1000 0000 = **1**000 0000

③左移7位：

**1**000 0000 << 7 = 1111 111**1**

④和0x01与运算：

1111 111**1** & 0000 0001 = 0000 000**1**

此时，取出ch[0]的ascii码的第2位。因此，temp[8\*m+n]=temp[1]的ascii码为0000 0001，

即temp[1]=1。

代码：

for(m=0; m<taille\_message; m++)

{

for(n=0; n<8; n++)

temp[8\*m+n]=((0x80&(ch[m]<<n))>>7)&0x01;

}

1. 将“清零操作”之后的像素点和“按位分离”的数组temp[ ]的各元素相加，即可完成加密。

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

1. 由于加密不一定需要所有的像素，因此“清零操作”之前，先根据ch[ ]的长度，计算出哪些像素是需要被清零的。

我们按照“先行后列”的规则进行加密，则设置如下两个常量：

I=(taille\_message\*8)/image->width;

J=(taille\_message\*8)%image->width;

将加密分为两个部分进行：

（1）0≤i<I，0≤j<image->width

1. i=I，0≤j<J

对于每个像素image->data[i][j]，加在其上的code对应temp[k]=temp[(image->width)\*i+j]。

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=(taille\_message\*8)/image->width;

J=(taille\_message\*8)%image->width;

Le codage est divisé en deux parties:

（1）0≤i<I，0≤j<image->width

（2）i=I，0≤j<J

Pour chaque pixel image->data[i][j], le code qui lui est ajouté correspond à temp[k] =temp[(image->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 |

