

**Федеральное государственное бюджетное образовательное
учреждение высшего образования “Московский
политехнический университет”**

Факультет информационных технологий

Кафедра “Информационная безопасность”

КУРСОВАЯ РАБОТА

По дисциплине: “Технологии и методы программирования”

На тему: “Разработка программного обеспечения для внедрения
скрытой информации в растровые изображения”

Выполнил:

студент 1 курса

группы 191-331,

Беляков Максим Сергеевич

Проверил:

Харченко Е.А

Москва, 2020

Содержание

Содержание	2
Введение	3
Глава 1. Описание алгоритма LSB Replacement.....	4
Глава 2. Реализация алгоритма	6
2.1 Алгоритм работы приложения	6
2.2 ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ, КОНСТРУКТОРА И ВМР ХЕДЕРА	7
2.3 ФУНКЦИИ.....	8
Глава 3. Использование приложения	12
Заключение	14
Список используемой литературы	15

Введение

Стеганография — это междисциплинарная наука и искусство передавать сокрытые данные, внутри других, не сокрытых данных.

Секретную информацию можно записать в метаданные файла или же напрямую в основное содержимое. Возьмем, например, картинку. С точки зрения компьютера она представляет собой набор из сотен тысяч точек-пикселей. У каждого пикселя есть «описание» — информация о его цвете.

Для формата RGB, который используется в большинстве цветных картинок, это описание занимает в памяти 24 бита. Если в описании некоторых или даже всех точек 1–3 бита будет занято секретной информацией, на картинке в целом изменения будут неразличимы. А за счет огромного числа пикселей всего в изображение вписать можно довольно много данных.

В большинстве случаев прячут информацию в пиксели и извлекают ее оттуда при помощи специальных утилит. Иногда для этой цели пишут собственные скрипты или добавляют нужную функциональность в программы другого назначения. А иногда пользуются готовыми кодами, которых в сети немало.

Учитывая то, как быстро развиваются технологии, в том числе и технологии злоумышленников, нельзя не заметить то, что с каждым днем все сложнее создать пароль и криптографические ключи, которые нельзя было бы взломать, поэтому стеганография приобретает сильную актуальность, в связи с тем, что если злоумышленник не поймет, что материал, что попал к нему в руки является засекреченным, то и взламывать не станет.

Глава 1. Описание алгоритма LSB Replacement

LSB Replacement - простейший метод внедрения скрытой информации в цифровые изображения, манипулирующие младшими битами значений. Бит внедряемого сообщения оказывается в некоторых случайных младших битах изображения-контейнера.

Общий принцип подобных методов заключается в замене избыточной, малозначимой части изображения битами секретного сообщения. То есть на глаз невозможно будет определить, поменялось ли что-нибудь в изображении или нет. Для того, чтобы извлечь само сообщение, необходимо знать, по какому алгоритму составляющие его биты размещались по контейнеру.

Метод, что использовался в этой работе, LSB Replacement, состоит в простой замене на бит внедряемого сообщения.

В данном инженерном проекте, формат изображений, с которым мы будем работать, является - BMP.

Пиксел изображения в 24-х битном. BMP формате занимает 3 Байта (24 бита) памяти (соответственно по 1 байту на каждый канал - Red, Green, Blue (RGB))

Один символ текста занимает 1 Байт (8 бит) дискового пространства.

Если заменять Байт изображения Байтом текста, то мы получим абсолютно другой цвет пиксела, и, как следствие, сильное искажение изображения.

Поэтому наиболее удобно заменять только 1 бит одного из каналов Пиксела на 1 бит Текста. Такая подмена будет незаметна для человеческого глаза. (При желании можно заменять последний бит КАЖДОГО из каналов... Алгоритм изменится незначительно).

Таким образом, один символ текста будет кодироваться в 8 пикселей изображения.

Для того, чтобы понять, какой бит будем занимать, следует обратиться к примеру.

Возьмем пиксель со значением 255

В двоичном представлении он будет иметь вид

11111111

Если заменить последний бит

11111110

Он приобретет следующее десятичное значение - 127

Разница на лицо. Но если заменить первый бит

01111111

Приобретает значение 254. То есть разница невелика. Исходя из этого заключения мы будем менять именно первый бит у всех пикселей

Глава 2. Реализация алгоритма

2.1 Алгоритм работы приложения

Для того, чтобы реализовать данный алгоритм сначала надо его разбить на несколько действий

1. Загружаем изображение
2. Переносим в массив
3. Вводим секретное сообщение
4. Шифруем изображение:
5. Сохраняем зашифрованное изображение, для наглядности
6. Дешифруем изображение
7. Выводим секретное сообщение

Реализовывать данный алгоритм будем на языке C++ в Visual Studio 2017

2.2 ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ, КОНСТРУКТОРА И ВМР ХЕДЕРА

1. Вынесем в отдельный файл структуру заголовку BMP файла

```
stegan.h
1  typedef unsigned __int16 WORD;
2
3  typedef struct {
4      WORD  bfType;
5      int   bfSize;
6      int   bfReserved;
7      int   bfOffBits;
8
9      int   biSize;
10
11
12      int   biWidth;
13      int   biHeight;
14      WORD  biPlanes;
15      WORD  biBitCount;
16      int   biCompression;
17
18
19      int   biSizeImage;
20
21      int   biXPelsPerMeter;
22      int   biYPelsPerMeter;
23      int   biClrUsed;
24
25      int   biClrImportant;
26
27  } BMPheader;
```

2. Пропишем начало класса и глобальные переменные, которые используем в будущем

```
class Stegan {
    vector<int> key_main;
    int times_in;
    int times_all;
    string text;
    int _mx, _my;
};
```

3. Пропишем конструктор класса, который каждый раз, как создается объект будет инициализировать наши шаги

```
public:
    Stegan() {
        times_in = 0;
        times_all = 0;
    }
```

2.3 ФУНКЦИИ

1. Начинаем писать метод загрузки изображения. Напишем для начала непосредственно загрузку хедера и проверки на правильность изображения

```
int *loadBMP(const char *fname)
{
    int mx, my = -1;
    FILE *f = fopen(fname, "rb");
    if (!f) return NULL;
    BMPheader bh;
    size_t res;

    // читаем заголовок
    res = fread(&bh, 1, sizeof(BMPheader), f);
    if (res != sizeof(BMPheader)) { fclose(f); return NULL; }

    // проверяем сигнатуру
    if (bh.bfType != 0x4d42 && bh.bfType != 0x4349 && bh.bfType != 0x5450) { fclose(f); return NULL; }

    // проверка размера файла
    fseek(f, 0, SEEK_END);
    int filesize = ftell(f);
    // восстановим указатель в файле:
    fseek(f, sizeof(BMPheader), SEEK_SET);
    // проверим условия
    if (bh.bfSize != filesize ||
        bh.bfReserved != 0 ||
        bh.biPlanes != 1 ||
        (bh.biSize != 40 && bh.biSize != 108 && bh.biSize != 124) ||
        bh.bfOffBits != 14 + bh.biSize ||

        bh.biwidth < 1 || bh.biwidth > 10000 ||
        bh.biHeight < 1 || bh.biHeight > 10000 ||
        bh.biBitCount != 24 || // пока рассматриваем только полноцветные изображения
        bh.biCompression != 0 // пока рассматриваем только несжатие изображения
    )
    {
        fclose(f);
        return NULL;
    }
    // Заголовок прочитан и проверен, тип - верный (BGR-24), размеры (mx, my) найдены
```


2. Загоним данные в массив

```
mx = bh.biWidth;
_myx = mx;
my = bh.biHeight;
_my = my;
int mx3 = (3 * mx + 3) & (-4); // Compute row width in file, including padding to 4-byte boundary
unsigned char *tmp_buf = new unsigned char[mx3*my]; // читаем данные
res = fread(tmp_buf, 1, mx3*my, f);
if ((int)res != mx3 * my) { delete[]tmp_buf; fclose(f); return NULL; }
// данные прочитаны
fclose(f);
// выделим память для результата
int * v = new int[mx*my];
// Перенос данных (не забудем про BGR->RGB)
unsigned char *ptr = (unsigned char *)v;
for (int y = my - 1; y >= 0; y--) {
    unsigned char *pRow = tmp_buf + mx3 * y;
    for (int x = 0; x < mx; x++) {
        *ptr++ = *(pRow + 2);
        *ptr++ = *(pRow + 1);
        *ptr++ = *pRow;
        pRow += 3;
        ptr++;
    }
}
delete[]tmp_buf;
return v; // OK
}
```

3. Реализуем две функции, которые очень помогут при шифровании и дешифровании изображения

```
int GetBitValue(unsigned char &B, int N)
/* Получает значение N-ого бита в байте B
   Возвращает 1 или 0 (в зависимости от значения Бита)
*/
{
    int k = 256;

    for (int i = 0; i < N; i++) k /= 2;

    if ((B & k) != 0) return 1;
    else return 0;
}

unsigned char ReadBitToByte(int Bit, unsigned char &B)
/* Записывает в байт B на последнюю позицию бит Bit
*/
{
    int A = 1;
    unsigned char Result = B;

    if (Bit == GetBitValue(B, 8)) return B;
    else if (Bit == 1) return Result = Result & A;
    else if (Bit == 0) return B ^ A;

    return NULL;
}
```

4. Реализуем метод шифрования изображения

```
int *crypt(int*v, int key, int rate) {  
  
    int all_pixels = _mx * _my;  
    int i = 0;  
    unsigned char *ptr = (unsigned char*)v;  
    int mx3 = (3 * _mx + 3) & (-4);  
    unsigned char *tmp_buf = new unsigned char[mx3*_my];  
    for (int y = _my - 1; y >= 0; y--) { //идем по массиву  
        unsigned char *pRow = tmp_buf + mx3 * y;  
        for (int x = 0; x < _mx; x++) {  
  
            if (i == rate && key>0) { //условие выполняется, если ключ больше 0 и момент рэйта  
  
                *ptr++ = ReadBitToByte(key%10, *pRow); //последний цифру ключа записываем в пиксель  
                i = 0;  
                key = key / 10;  
                times_in++; //количество раз, которое зайдет алгоритм  
            }  
            i++;  
            pRow += 3;  
            ptr++;  
        }  
    }  
    times_all = times_in; //такой же счетчик, как и times_in. Просто не обнуляется и нужен только для вывода количества обходов алгоритма  
    delete[] tmp_buf;  
    return v;  
}
```

5. Реализуем метод дешифрования изображения

```
int *de_crypt(int*v, int rate) {  
    int i = 0;  
    unsigned char *ptr = (unsigned char*)v;  
    int mx3 = (3 * _mx + 3) & (-4);  
    unsigned char *tmp_buf = new unsigned char[mx3*_my];  
    for (int y = _my - 1; y >= 0; y--) {  
        unsigned char *pRow = tmp_buf + mx3 * y;  
        for (int x = 0; x < _mx; x++) {  
  
            if (i == rate && times_in>0) { //выполнится столько раз, сколько заходил в предыдущие разы алгоритм  
                int bit = GetBitValue(*ptr++, 8); //смотрим 8 бит  
                times_in--; //количество раз отнимается  
                i = 0;  
                key_main.push_back(bit); //заносим бит в вектор ключа  
            }  
            i++;  
            pRow += 3;  
            ptr++;  
        }  
    }  
    delete[] tmp_buf;  
    return v;  
}
```

6. Реализуем два геттера для вывода зашифрованного сообщения и шага

```
void get_message() { //вывести вектор сообщения  
    cout << "THE SECRET KEY IS - " << " ";  
    for (int i = key_main.size()-1; i >= 0; i--) { //от последнего члена к первому, чтобы получилось также, как и ключ был изначально  
        cout << key_main[i] << " ";  
    }  
    cout << endl;  
}  
  
void get_time() {  
    cout << "THERE WERE TIMES - " << times_all << endl;  
}
```

7. Реализуем последний метод - сохранения изображения

```
int saveBMP(const char *fname, int *v) // В каждом элементе упаковано все три RGB-байта
{
    BMPheader bh; // Заголовок файла, sizeof(BMPheader) = 56
    memset(&bh, 0, sizeof(bh));
    bh.bfType = 0x4d42; // 'BM'
    // Найдем длину строки в файле, включая округление вверх до кратного 4:
    int mx3 = (3 * _mx + 3) & (-4);
    int filesize = 54 + _my * mx3;
    bh.bfSize = filesize;
    bh.bfReserved = 0;
    bh.biPlanes = 1;
    bh.biSize = 40;
    bh.bfOffBits = 14 + bh.biSize;
    bh.biWidth = _mx;
    bh.biHeight = _my;
    bh.biBitCount = 24;
    bh.biCompression = 0;

    FILE *f = fopen(fname, "wb");
    if (!f) return -1;
    size_t res;

    // пишем заголовок
    res = fwrite(&bh, 1, sizeof(BMPheader), f);
    if (res != sizeof(BMPheader)) { fclose(f); return -1; }

    // подготовим временный буфер
    unsigned char *tmp_buf = new unsigned char[mx3*_my];
    // Перенос данных (не забудем про RGB->BGR)
    unsigned char *ptr = (unsigned char *)v;
    for (int y = _my - 1; y >= 0; y--) {
        unsigned char *pRow = tmp_buf + mx3 * y;
        for (int x = 0; x < _mx; x++) {
            *(pRow + 2) = *ptr++;
            *(pRow + 1) = *ptr++;
            *pRow = *ptr++;
            pRow += 3;
            ptr++;
        }
    }
    // сбросим в файл
    fwrite(tmp_buf, 1, mx3*_my, f);
    fclose(f);
    delete[] tmp_buf;
    return 0; // ОК
}
```

Глава 3. Использование приложения

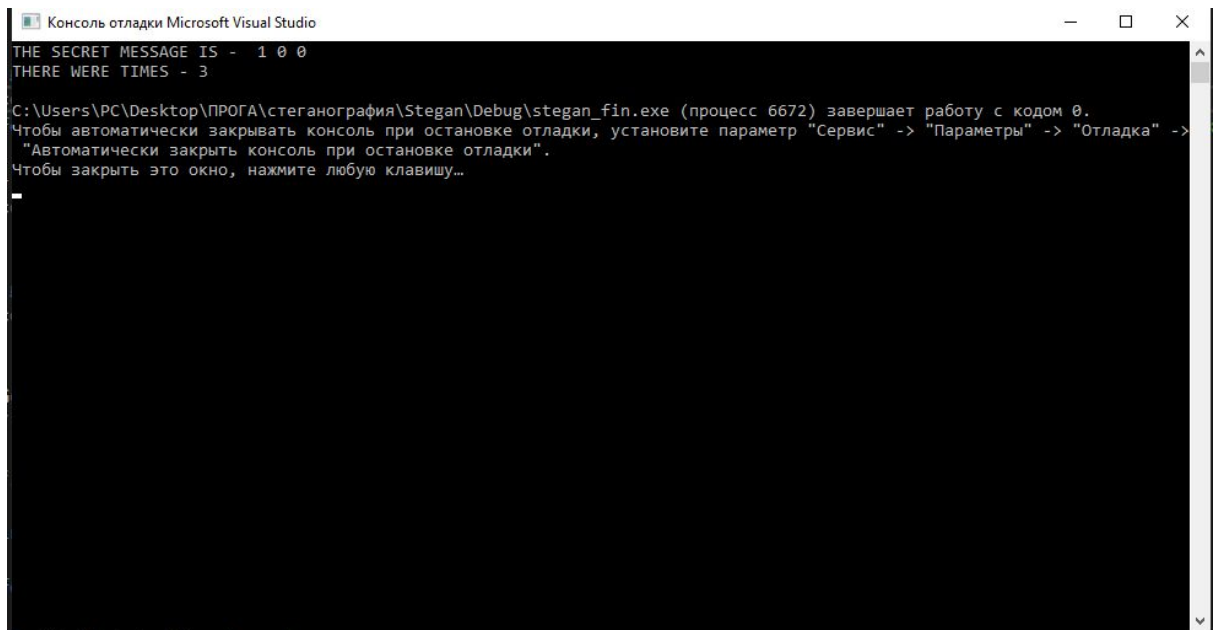
Код реализован и готов к использованию. Осталось только прописать код `main()` и показать, что непосредственно выведет само консольное приложение

1. Код в `main()`

```
int main() {
    int key = 100; //создали ключ
    int rate = 124; //создали рэйт
    Stegan st1; //создали объект
    int * v = st1.loadBMP("C:/Users/PC/Desktop/ПРОГА/rainbow.bmp"); //загрузили bmp файл
    int *modified_v = st1.crypt(v, key, rate); //зашифровали bmp файл
    st1.de_crypt(modified_v, rate); //расшифровали bmp файл
    st1.get_message(); //вывели сообщение
    st1.get_time(); //вывели количество раз
    st1.saveBMP("C:/Users/PC/Desktop/ПРОГА/rainbow2.bmp", modified_v); //сохранили зашифрованный bmp файл

    delete[]modified_v;
}
```

2. Консольный отладчик

The image shows a screenshot of the 'Консоль отладки Microsoft Visual Studio' (Visual Studio Debug Console) window. The window has a title bar with standard Windows window controls (minimize, maximize, close). The console output is as follows:
THE SECRET MESSAGE IS - 1 0 0
THERE WERE TIMES - 3
C:\Users\PC\Desktop\ПРОГА\стеганография\Stegan\Debug\stegan_fin.exe (процесс 6672) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
The console text is displayed in a monospaced font on a black background with white text. The window has a vertical scrollbar on the right side.

Как видно в консольном отладчике и сравнивая данные из части `main()`, видно, что программа успешно отработала и вывела правильные данные

3. Изображение, с которым проводилась отладка приложения



Заключение

Стеганография - важная часть информационной безопасности. Несмотря на доказанную эффективность, не стоит применять стеганографию как единственный метод и способ шифровки информации. Коэффициент полезного действия шифрования информации повышается многократно, если использовать комплексный метод, включающий в себя не только стеганографию, но и криптографию.

Приложение, что было написано в ходе данной курсовой работы полностью отлажено и готово к выпуску в промышленном формате. Наибольшую эффективность данного приложения можно раскрыть в следующих типах: мессенджеры, электронная почта, приложения для частных компаний.

Список используемой литературы

1. Рябко Б.Я., Фионов А.Н “Основы современной криптографии и стеганографии.” - М.: Горячая линия - Телеком, 2016.
2. BMP
<https://ru.wikipedia.org/wiki/BMP>