

# **Отчёт по лабораторной работе №7**

**Дисциплина: Информационная безопасность**

Выполнил: Танрибергенов Эльдар

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задания</b>	<b>5</b>
<b>3</b>	<b>Теоретические сведения</b>	<b>6</b>
<b>4</b>	<b>Выполнение работы</b>	<b>8</b>
<b>5</b>	<b>Выводы</b>	<b>14</b>
<b>6</b>	<b>Ответы на контрольные вопросы</b>	<b>15</b>

# Список иллюстраций

4.1	Функция кодирования текста в шестнадцатеричный код . . . . .	8
4.2	Функция декодирования шестнадцатеричного кода в текст . . . . .	9
4.3	Функция однократного гаммирования . . . . .	9
4.4	Функция однократного гаммирования . . . . .	10
4.5	Функция генерации ключа . . . . .	11
4.6	Запускающая программу функция . . . . .	11
4.7	Действия программы . . . . .	12
4.8	Результат . . . . .	12

# **1 Цель работы**

Освоить на практике применение режима однократного гаммирования.

## 2 Задания

- Требуется разработать приложение, позволяющее шифровать и дешифровать данные в режиме однократного гаммирования. Приложение должно:
  1. Определить вид шифротекста при известном ключе и известном открытом тексте.
  2. Определить ключ, с помощью которого шифротекст может быть преобразован в некоторый фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста.

### 3 Теоретические сведения

Предложенная Г. С. Вернамом так называемая «схема однократного использования (гаммирования)» является простой, но надёжной схемой шифрования данных. Гаммирование представляет собой наложение (снятие) на открытые (зашифрованные) данные последовательности элементов других данных, полученной с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных. Иными словами, наложение гаммы — это сложение её элементов с элементами открытого (закрытого) текста по некоторому фиксированному модулю, значение которого представляет собой известную часть алгоритма шифрования. В соответствии с теорией криптоанализа, если в методе шифрования используется однократная вероятностная гамма (однократное гаммирование) той же длины, что и подлежащий сокрытию текст, то текст нельзя раскрыть. Даже при раскрытии части последовательности гаммы нельзя получить информацию о всём скрываемом тексте. Наложение гаммы по сути представляет собой выполнение операции сложения по модулю 2 (XOR) между элементами гаммы и элементами подлежащего сокрытию текста. Такой метод шифрования является симметричным, так как двойное прибавление одной и той же величины по модулю 2 восстанавливает исходное значение, а шифрование и расшифрование выполняется одной и той же программой. Открытый текст имеет символьный вид, а ключ — шестнадцатеричное представление. Ключ также можно представить в символьном виде, воспользовавшись таблицей ASCII-кодов. К. Шеннон доказал абсолютную стойкость шифра в случае, когда однократно используемый ключ, длиной, равной длине исходного

сообщения, является фрагментом истинно случайной двоичной последовательности с равномерным законом распределения. Криптоалгоритм не даёт никакой информации об открытом тексте: при известном зашифрованном сообщении  $C$  все различные ключевые последовательности  $K$  возможны и равновероятны, а значит, возможны и любые сообщения  $P$ . Необходимые и достаточные условия абсолютной стойкости шифра: – полная случайность ключа; – равенство длин ключа и открытого текста; – однократное использование ключа.

## 4 Выполнение работы

Программа написана на языке программирования C++

```
// Кодирует текст в шестнадцатеричную последовательность
string encode_hex(string norm_txt){
    string hex_txt;
    stringstream ss;
    string::const_iterator cii;
    unsigned char c;
    int cnum;
    char *chr_norm_txt = new char[norm_txt.length()];

    for(cii=norm_txt.begin(); cii!=norm_txt.end(); cii++){
        c = *cii;
        cnum = int(c);
        ss << hex << cnum;
    }
    hex_txt = ss.str();
    return hex_txt;
}
```

Рис. 4.1: Функция кодирования текста в шестнадцатеричный код

В цикле посимвольно разбирается строка. Конвертируется в целочисленный вид, а далее - в шестнадцатеричный. После 16-ричные числа записываются в строковый поток. Полученная строка возвращается функцией.



```

// Декодирует шестнадцатеричную последовательность в текст
string decode_hex(string hex_txt){
    string norm_txt, tmp="";
    stringstream ss;
    string::const_iterator cii;
    char *chr_hex_txt = new char[hex_txt.length()];
    int n=0;

    for(cii=hex_txt.begin(); cii!=hex_txt.end(); cii++){
        tmp += *cii;
        chr_hex_txt[n] = *cii;
        n++;
    }
    hex_txt = tmp;

    for(int i=0; i<n; i+=2){
        tmp="";
        for(int j=i; j<i+2; j++)
            tmp += chr_hex_txt[j];
        ss << char(stoi(tmp,nullptr,16));
    }
    norm_txt = ss.str();

    delete [] chr_hex_txt;
    chr_hex_txt = nullptr;

    return norm_txt;
}

```

Рис. 4.2: Функция декодирования шестнадцатеричного кода в текст

У меня не получилось нормально работать с классом строк C++, поэтому я скопировал значения строки в массив символов. Далее по 2 символа (16-ричное число) превращал в целое число и добавлял в строковый поток их конвертированное в символ значение. Далее функция возвращает получившуюся строку.

```

// Гаммирование
string one_time_gamming(string hex_txt, string key){
    string gammed_txt="", str_xor, hex1, hex2;
    stringstream ss;
    bitset<8> bin_xor;
    int int_xor, int_sumnd1, int_sumnd2;
    string::const_iterator cii, cij;
    char *chr_message = new char[key.length()];
    char *chr_key = new char[key.length()];
    int n=0, m=0;

    for (cii=hex_txt.begin(); cii!=hex_txt.end(); cii++){
        chr_message[n] = *cii;
        n++;
    }
    for (cij=key.begin(); cij!=key.end(); cij++){
        chr_key[m] = *cij;
        m++;
    }
}

```

Рис. 4.3: Функция однократного гаммирования

Здесь в символьные массивы занесены данные строк.

```
for(int i=0; i<n; i+=2){
    hex1 = "";
    hex2 = "";
    for(int j=i; j<i+2; j++)
    {
        hex1 += chr_message[j];
        hex2 += chr_key[j];
    }
    int_sumnd1 = stoi(hex1, nullptr, 16);
    int_sumnd2 = stoi(hex2, nullptr, 16);
    bin_xor = bitset<8>(int_sumnd1) ^= bitset<8>(int_sumnd2);
    str_xor = bin_xor.to_string();
    int_xor = stoi(str_xor, nullptr, 2);
    if(int_xor < 16)
        ss << hex << 0 << int_xor;
    else
        ss << hex << int_xor;
}
gammed_txt = ss.str();

delete [] chr_message;
delete [] chr_key;
chr_message = nullptr;
chr_key = nullptr;

return gammed_txt;
}
```

Рис. 4.4: Функция однократного гаммирования

А здесь уже происходит суммирование по модулю 2: 16-ричное число превращается в 10-чное, затем 10-чное число превращается в двоичную последовательность `bitset`, и после этого двоичные значения проходят операцию `xor`. Далее значение вновь конвертируется в 10-чное и проходит проверку. Если меньше 16, то надо дописать 0, т.к. система этого не делает. После всего этого значение в 16-чном виде заносится в строковый поток. Функция возвращает полученную строку.

```
// Генерация ключа

string key_gen(string text, int rand_num){

    string key;
    stringstream ss;
    string::const_iterator cii;

    for(cii=text.begin(); cii!=text.end(); cii++)
        ss << hex << (rand()%100 + rand_num)%16;S
    key = ss.str();

    return key;
}
```

Рис. 4.5: Функция генерации ключа

Здесь просто цикл до размера шифротекста, в котором в строковый поток добавляется случайное число от 0 до 15 в 16-ричном виде. И строка возвращается функцией.

```
#include <iostream>
#include <string>
#include "functions.h"
using namespace std;

int main(){

    srand(time(NULL));
    string open_txt, hex_txt, key, encrypted_txt, decrypted_txt, decoded_txt;

    open_txt = "Yes";
    cout << "\n Исходное сообщение:\n\n " << open_txt << endl;
    hex_txt = encode_hex(open_txt);
    cout << " " << hex_txt << endl;

    key = key_gen(hex_txt,rand()%10);
    cout << "\n Ключ:\n " << key << endl;

    encrypted_txt = one_time_gamming(hex_txt, key);
    cout << "\n Зашифрованный текст:\n " << encrypted_txt << endl;
```

Рис. 4.6: Запускающая программу функция

Это уже главная функция (“запускающая”). Здесь просто выводится на консоль сообщения и вызываются функции из отдельного файла.

```

key = key_gen(encrypted_txt, rand()%10);

cout << "\n Расшифровка текста...\n\n Пробный ключ:\n " << key << endl;
decrypted_txt = one_time_gamming(encrypted_txt, key);
decoded_txt = decode_hex(decrypted_txt);
cout << "\n Расшифрованное сообщение:\n\n " << decoded_txt << endl << endl;

cout << " Подбор ключа..." << endl;

do{
    key = key_gen(encrypted_txt, rand()%10);
    decrypted_txt = one_time_gamming(encrypted_txt, key);
    decoded_txt = decode_hex(decrypted_txt);

}while(decoded_txt!=open_txt);

cout << "\n Ключ:\n " << key << endl;
cout << "\n " << decoded_txt << endl << endl;

return 0;
}

```

Рис. 4.7: Действия программы

Далее происходит подбор ключа: ключ генерируется до тех пор, пока, расшифрованная с его помощью строка, не будет равна исходному открытому тексту.

```

etanribergenov@etanribergenov:~/code
File Edit View Search Terminal Help
[etanribergenov@etanribergenov code]$ g++ lab07.cpp -o lab07
[etanribergenov@etanribergenov code]$ ./lab07

Исходное сообщение:
Yes
596573

Ключ:
1cd965

Зашифрованный текст:
45bc16

Расшифровка текста...

Пробный ключ:
8fa7fd

Расшифрованное сообщение:
0000

Подбор ключа...

Ключ:
1cd965

Yes
[etanribergenov@etanribergenov code]$

```

Рис. 4.8: Результат

Результат работы программы. Выбрал короткое слово “Yes”, потому что иначе подбор происходит очень долго.

## **5 Выводы**

В результате выполнения работы я освоил на практике применение режима однократного гаммирования.

## 6 Ответы на контрольные вопросы

1. Однократное гаммирование - это суммирование по модулю 2 16-ричных последовательностей.
2. Необходимость иметь большие объёмы данных для использования в качестве гаммы. Угроза криптоанализа. Защита может быть нарушена, если злоумышленник получит достаточно большое количество зашифрованных сообщений и соответствующих ключевых потоков гаммы. Неустойчивость к ошибкам передачи данных. Для этого метода необходимы дополнительные механизмы проверки целостности и коррекции ошибок. Уязвимость к повторному использованию гаммы. Если гамма будет повторно использована, то система станет уязвимой, и её защита будет нарушена.
3. Высокая степень защиты от криптоанализа методом подбора ключа. Возможность обрабатывать данные любой длины. Отсутствие расширения сообщения. Неразличимость шифрованного текста с исходным. Если гамма является случайной и используется только один раз, то шифрованный текст будет неразличим с исходным текстом без знания ключа.
4. Потому что должен быть выделен одинаковый объём памяти.
5. В однократном гаммировании используется операция сложения по модулю 2 (XOR).
6. Сложить по модулю 2 байты (гаммировать).
7. Выполнить гаммирование открытого текста и шифротекста.
8. Полная случайность ключа; равенство длин ключа и открытого текста; однократное использование ключа.