Отчет по лабораторной работе №7

Элементы криптографии. Однократное гаммирование

Поленикова Анна Алексеевна

Содержание

Список иллюстраций

Цель работы

Освоить на практике применение режима однократного гаммирования.

Теоретическая справка

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) (обозначаемая знаком $\oplus$) между элементами гаммы и элементами подлежащего сокрытию текста. Напомним, как работает операция XOR над битами: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$. Такой метод шифрования является симметричным, так как двойное прибавление одной и той же величины по модулю 2 восстанавливает исходное

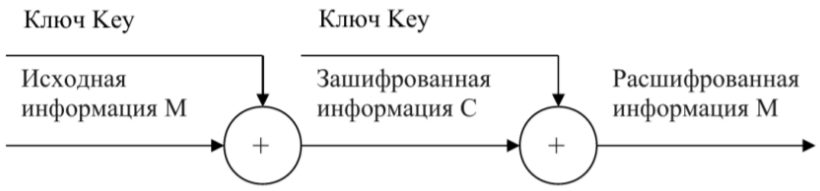

Рис. 0.1: Схема однократного использования Вернама

значение, а шифрование и расшифрование выполняется одной и той же программой. Если известны ключ и открытый текст, то задача нахождения шифротекста заключается в применении к каждому символу открытого текста следующего правила:

$$C_i = P_i \oplus K_i$$

, где C_i — i-й символ получившегося зашифрованного послания, P_i — i-й символ открытого текста, K_i — i-й символ ключа, $i = 1, m$. Размерности открытого текста и ключа должны совпадать, и полученный шифротекст будет такой же длины. Если известны шифротекст и открытый текст, то задача нахождения ключа решается также в соответствии с (7.1), а именно, обе части равенства необходимо сложить по модулю 2 с P_i:

$$C_i \oplus P_i = P_i \oplus K_i \oplus P_i = K_i,$$

$$K_i = C_i \oplus P_i$$

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.

Выполнение лабораторной работы

1. Написала следующую программу на C++, шифрующую введенное сообщение при помощи случайно сгенерированного ключа.

```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

string tranform(string toEncrypt) {
    string transformedMessage;
    for (int i = 0; i < toEncrypt.size(); ++i) {
        int ia = toascii(toEncrypt.at(i));
        transformedMessage.push_back(ia);
    }
    return transformedMessage;
}

string keyGeneration(string toEncrypt) {
    string key;
    static const char code[] = "0123456789";

    for (int i = 0; i < toEncrypt.size(); ++i) {
        key += code[rand() % (sizeof(code) - 1)];
    }

    return key;
}

string encryptDecrypt(string toEncrypt, string key) {
    string output = toEncrypt;

    for (int i = 0; i < toEncrypt.size(); i++)
        output[i] = toEncrypt[i] ^ key[i];

    return output;
}
```

Рис. 0.1: Функции программы-шифратора

```cpp
int main(int argc, const char* argv[])
{
    setlocale(LC_ALL, "Russian");

    string message = "С Новым Годом, друзья!";

    string transformedMessage = tranform(message);

    string key = keyGeneration(transformedMessage);

    string encrypted = encryptDecrypt(transformedMessage, key);


    cout << "Сообщение: " << message << "\n";
    cout << "Преобразованное сообщение: ";
    for (int i = 0; i < transformedMessage.size(); ++i) {
        cout << int(transformedMessage[i]);
    }
    cout << "\n" << "Ключ: " << key << "\n";
    cout << "Зашифрованное сообщение: ";
    for (int i = 0; i < encrypted.size(); ++i) {
        cout << int(encrypted[i]);
    }

    return 0;
}
```

Рис. 0.2: main функция программы-шифратора

2. Результат работы программы можно увидеть ниже:

```
Консоль отладки Microsoft Visual Studio
Сообщение: С Новым Годом, друзья!
Преобразованное сообщение: 8132771109812310832671101001101084432100112115103124127 33
Ключ: 174094882455171152761 4
Зашифрованное сообщение: 962312194917984241139081919327178569658074782 1
```

Рис. 0.3: Результат работы программы-шифратора

Вывод

В ходе выполнения лабораторной работы было освоено на практике применение режима однократного гаммирования.