Л.7. Элементы криптографии. Однократное гаммирование

Греков Максим Сергеевич

2021

RUDN University, Moscow, Russian Federation

Цель работы

Цель работы

Освоить на практике применение режима однократного гаммирования

Задание

Задание

Нужно подобрать ключ, чтобы получить сообщение «С Новым Годом, друзья!». 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.

Теория

Гаммирование

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.

Как применять

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 между элементами гаммы и элементами подлежащего сокрытию текста.

Нахождение шифротекста

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. Размерности открытого текста и ключа должны совпадать, и полученный шифротекст будет такой же длины.

Нахождение ключа

Если известны шифротекст и открытый текст, то, чтобы найти ключ, обе части равенства необходимо сложить по модулю 2 с P_i:

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

$$K_i = C_i \oplus P_i$$

Стойкость шифра

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.

Условия стойкости шифра

Необходимые и достаточные условия абсолютной стойкости шифра:

- полная случайность ключа;
- равенство длин ключа и открытого текста;
- однократное использование ключа.

Ход работы

Класс Gumming

Для разработки приложения был описан класс *Gumming* (рис. 1), описывающий интересующие нас поля (открытый текст, закрытый текст, ключ шифрования), а также методы для работы с ними:

```
class Gumming { //гаммирование
private:
    string P; //открытый текст
    string C; //закрытый текст
    string K; //ключ шифрования
public:
    string getP() { return P; };
    string getC() { return C; };
    string getK() { return K; };

    void setP(string p) { P = p; };
    void setC(string c) { C = c; };
    void setK(string k) { K = k; };

    Gumming() {};
    Gumming(string p, string c, string k) { P = p; C = c; K = k; };

    void keygen(); //сгенерировать ключ
    void key(); //подобрать ключ
    void encrypt(); //зашифровать
    void decipher(); //расшифровать
};
```

Figure 1: Класс Gumming

Метод keygen()

Метод *keygen()* (рис. 2) позволяет сгенерировать псевдослучайный ключ такой же длины, как и открытый текст:

```
void Gumming::keygen() {
    K.clear();
    for (int i = 0; i < P.length(); i++)
        K += rand() % 256; //[0;255]
}
```

Figure 2: Метод keygen()

Метод key()

Метод *key()* (рис. 3) позволяет получать ключ, зная открытый и закрытый текст:

```
void Gumming::key() {
    if (P.length() != C.length()) {
        cout << "Несоответствие размерностей!" << endl;
        return;
    }
    K.clear();
    for (int i = 0; i < P.length(); i++)
        K += C[i] ^ P[i];
}
```

Figure 3: Метод key()

Метод encrypt()

Метод *encrypt()* (рис. 4) позволяет зашифровывать текст (получать закрытый текст), зная открытый текст и ключ:

```
void Gumming::encrypt() {
    if (P.length() != K.length()) {
        cout << "Несоответствие размерностей!" << endl;
        return;
    }
    C.clear();
    for (int i = 0; i < K.length(); i++)
        C += P[i] ^ K[i];
}
```

Figure 4: Метод encrypt()

Метод decipher()

Метод *decipher()* (рис. 5) позволяет расшифровывать текст (получать открытый текст), зная закрытый текст и ключ:

```
void Gumming::decipher() {
    if (C.length() != K.length()) {
        cout << "Несоответствие размерностей!" << endl;
        return;
    }
    P.clear();
    for (int i = 0; i < K.length(); i++)
        P += C[i] ^ K[i];
}
```

Figure 5: Метод decipher()

Вывод информации

Также были реализованы методы (рис. 6), служащие для вывода информации на экран в различных представлениях (текстовое, десятичное, шестнадцатеричное):

```
void printDec(const string s) {
    for (int i = 0; i < s.length(); i++)
        cout << dec << setw(4) << (int)(unsigned char)s[i];
    cout << endl;
}

void printHex(const string s) {
    for (int i = 0; i < s.length(); i++)
        cout << hex << setw(4) << (int)(unsigned char)s[i];
    cout << endl;
}

void printInfo(const string s) {
    cout << "Text: " << s << endl;
    cout << "Dec: ";
    printDec(s);
    cout << "Hex: ";
    printHex(s);
    cout << endl;
}
```

Figure 6: Вывод информации

Главная программа

В главной программе подключили кириллические символы в консоль (рис. 7), сгенерировали псевдослучайную последовательность, создали объект класса, установили открытый текст *“С Новым Годом, друзья!”*:

```
int main()
{
    setlocale(LC_ALL, "rus");
    srand(time(NULL));

    Gumming g;
    g.setP("С Новым Годом, друзья!");
    cout << "\tИсходное сообщение: " << endl;
    printInfo(g.getP());
```

Figure 7: Главная программа (1)

Главная программа

Затем вызвали методы для генерации псевдослучайного ключа и зашифровки текста (рис. 8). Всю информацию поэтапно выводим на экран:

```
g.keygen();
cout << "\tСгенерированный ключ: " << endl;
printInfo(g.getK());

g.encrypt();
cout << "\tЗашифрованное сообщение: " << endl;
printInfo(g.getC());
```

Figure 8: Главная программа (2)

Главная программа

После этого проверили работу методов для расшифровки и нахождения ключа (рис. 9), убедились в корректности работы программы по информации, получанной в консоли (рис. 10):

```
g.setP(""); //больше не знаем открытый
g.decipher();
cout << "\tРасшифрованное сообщение: " << endl;
printInfo(g.getP());

g.setK(""); //больше не знаем ключ
g.key();
cout << "\tНайденный ключ: " << endl;
printInfo(g.getK());
```

Figure 9: Главная программа (3)

Вывод программы

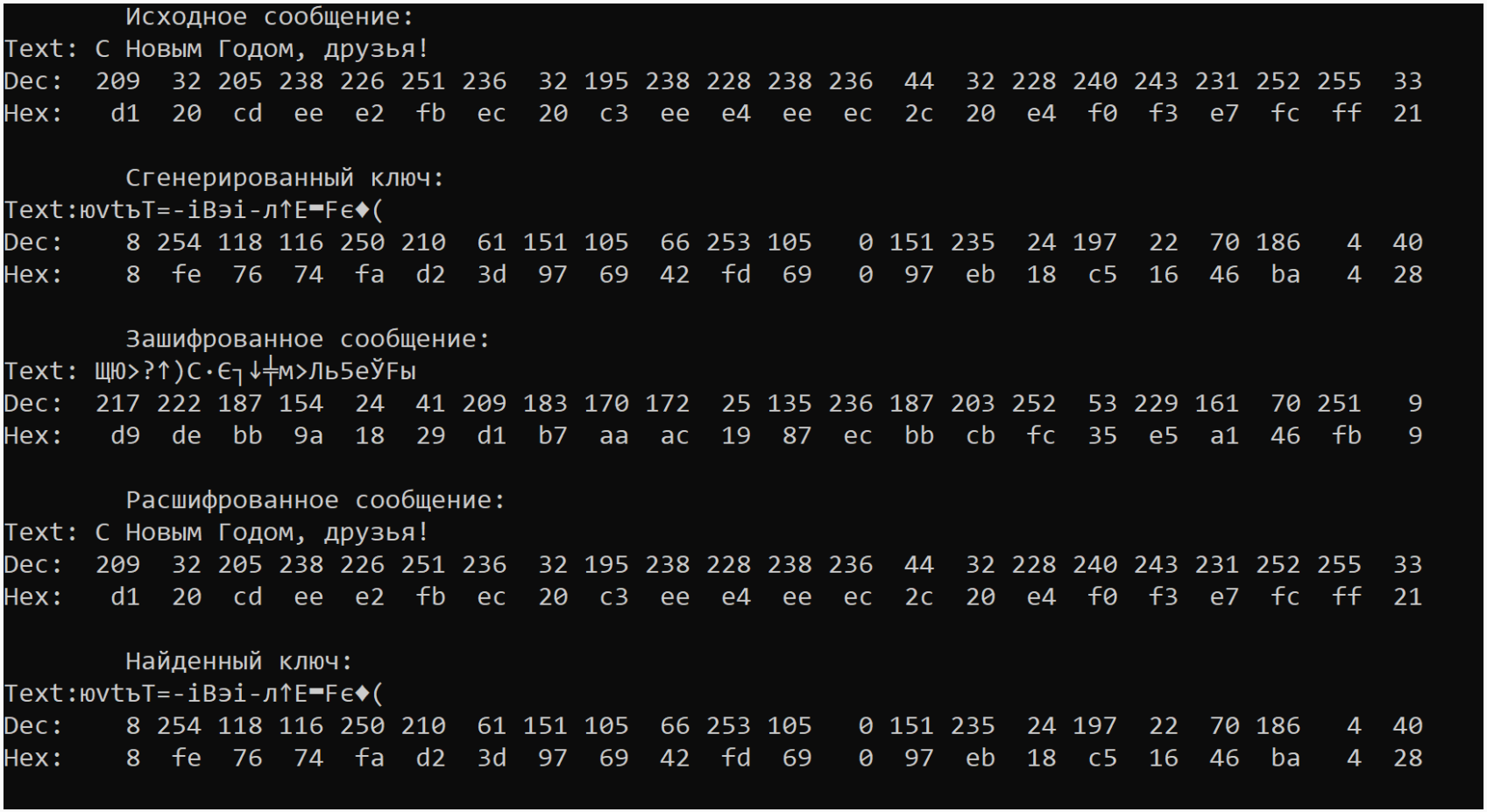

Figure 10: Вывод программы

Вывод

Вывод

Освоили на практике применение режима однократного гаммирования