

# **Информационная безопасность**

**Л.7. Элементы криптографии. Однократное гаммирование**

Греков Максим Сергеевич

2021

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теория</b>	<b>6</b>
3.1	Гаммирование . . . . .	6
3.2	Как применять . . . . .	6
3.3	Нахождение шифротекста . . . . .	7
3.4	Нахождение ключа . . . . .	7
3.5	Стойкость шифра . . . . .	7
3.6	Условия стойкости шифра . . . . .	8
<b>4</b>	<b>Ход работы</b>	<b>9</b>
4.1	Класс Gumming . . . . .	9
4.2	Метод keygen() . . . . .	9
4.3	Метод key() . . . . .	10
4.4	Метод encrypt() . . . . .	10
4.5	Метод decipher() . . . . .	11
4.6	Вывод информации . . . . .	11
4.7	Главная программа . . . . .	12
4.8	Вывод программы . . . . .	13
<b>5</b>	<b>Вывод</b>	<b>14</b>

# List of Figures

4.1	Класс Gumming . . . . .	9
4.2	Метод keygen() . . . . .	10
4.3	Метод key() . . . . .	10
4.4	Метод encrypt() . . . . .	10
4.5	Метод decipher() . . . . .	11
4.6	Вывод информации . . . . .	11
4.7	Главная программа (1) . . . . .	12
4.8	Главная программа (2) . . . . .	12
4.9	Главная программа (3) . . . . .	13
4.10	Вывод программы . . . . .	13

# 1 Цель работы

Освоить на практике применение режима однократного гаммирования

## 2 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.

## 3 Теория

### 3.1 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.

### 3.2 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.

### 3.3 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.

### 3.4 Нахождение ключа

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

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

$$K_i = C_i \oplus P_i$$

### 3.5 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  $P$ .

### 3.6 Условия стойкости шифра

Необходимые и достаточные условия абсолютной стойкости шифра:

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



## 4 Ход работы

### 4.1 Класс Gummig

Для разработки приложения был описан класс *Gummig* (рис. 4.1), описывающий интересующие нас поля (открытый текст, закрытый текст, ключ шифрования), а также методы для работы с ними:

```
class Gummig { //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; };

    Gummig() {};
    Gummig(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 4.1: Класс Gummig

### 4.2 Метод keygen()

Метод *keygen()* (рис. 4.2) 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 4.2: Метод keygen()

### 4.3 Метод key()

Метод *key()* (рис. 4.3) 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 4.3: Метод key()

### 4.4 Метод encrypt()

Метод *encrypt()* (рис. 4.4) 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 4.4: Метод encrypt()

## 4.5 Метод decipher()

Метод *decipher()* (рис. 4.5) 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 4.5: Метод decipher()

## 4.6 Вывод информации

Также были реализованы методы (рис. 4.6), 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 4.6: Вывод информации

## 4.7 Главная программа

В главной программе подключили кириллические символы в консоль (рис. 4.7), 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 4.7: Главная программа (1)

Затем вызвали методы для генерации псевдослучайного ключа и зашифровки текста (рис. 4.8). 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 4.8: Главная программа (2)

После этого проверили работу методов для расшифровки и нахождения ключа (рис. 4.9), убедились в корректности работы программы по информации, полученной в консоли (рис. 4.10):

```

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 4.9: Главная программа (3)

## 4.8 Вывод программы

```

Исходное сообщение:
Text: С Новым Годом, друзья!
Dec: 209 32 205 238 226 251 236 32 195 238 228 238 236 44 32 228 240 243 231 252 255 33
Hex: d1 20 cd ee e2 fb ec 20 c3 ee e4 ee ec 2c 20 e4 f0 f3 e7 fc ff 21

Сгенерированный ключ:
Text: юvтЪT=-iBэi-л†E=Fe♦(
Dec: 8 254 118 116 250 210 61 151 105 66 253 105 0 151 235 24 197 22 70 186 4 40
Hex: 8 fe 76 74 fa d2 3d 97 69 42 fd 69 0 97 eb 18 c5 16 46 ba 4 28

Зашифрованное сообщение:
Text: Щю>?†)С·Еγ ↓†м>ль5еЎҒы
Dec: 217 222 187 154 24 41 209 183 170 172 25 135 236 187 203 252 53 229 161 70 251 9
Hex: d9 de bb 9a 18 29 d1 b7 aa ac 19 87 ec bb cb fc 35 e5 a1 46 fb 9

Расшифрованное сообщение:
Text: С Новым Годом, друзья!
Dec: 209 32 205 238 226 251 236 32 195 238 228 238 236 44 32 228 240 243 231 252 255 33
Hex: d1 20 cd ee e2 fb ec 20 c3 ee e4 ee ec 2c 20 e4 f0 f3 e7 fc ff 21

Найденный ключ:
Text: юvтЪT=-iBэi-л†E=Fe♦(
Dec: 8 254 118 116 250 210 61 151 105 66 253 105 0 151 235 24 197 22 70 186 4 40
Hex: 8 fe 76 74 fa d2 3d 97 69 42 fd 69 0 97 eb 18 c5 16 46 ba 4 28

```

Figure 4.10: Вывод программы

## **5 Вывод**

Освоили на практике применение режима однократного гаммирования