Информационная безопасность

Л.8. Элементы криптографии. Шифрование (кодирование) различных исходных текстов одним ключом

Греков Максим Сергеевич

2021

Содержание

1 Цель работы **4**

2 Теория **5**
2.1 Однократное гаммирование одним ключом 5
2.2 Шифротексты телеграмм 5
2.3 Следствие свойства операции XOR 6
2.4 Получение второго открытого текста по первому 6

3 Ход работы **7**
3.1 Исходные данные 7
3.2 Код исходных данных 7
3.3 Функция tripl() 8
3.4 Последовательные вызовы tripl() 8
3.5 Последовательные открытия участков 10

4 Вывод **11**

List of Figures

2.1 Общая схема шифрования двух различных текстов одним ключом 5

3.1 Исходные данные 7
3.2 Функция tripl() 8
3.3 Последовательные вызовы функции *tripl()* с новыми данными . . 9
3.4 Последовательные открытия участков с новыми данными 10

1 Цель работы

Освоить на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом.

2 Теория

2.1 Однократное гаммирование одним ключом

Режим шифрования однократного гаммирования одним ключом двухвидов открытого текста реализуется в соответствии со схемой (рис. 2.1)

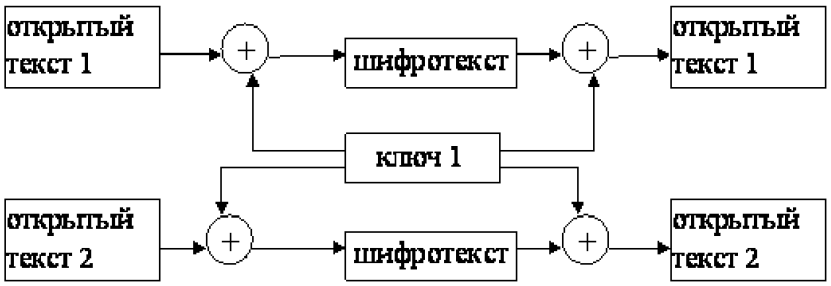

Figure 2.1: Общая схема шифрования двух различных текстов одним ключом

2.2 Шифротексты телеграмм

Шифротексты обеих телеграмм можно получить по формулам режима однократного гаммирования:

$$C_1 = P_1 \oplus K_i$$

$$C_2 = P_2 \oplus K_i$$

Открытый текст можно найти в соответствии с (рис. 2.1), зная шифротекст двух телеграмм, зашифрованных одним ключом.

2.3 Следствие свойства операции XOR

Для это оба равенства складываются по модулю 2. Тогда получаем:

$$C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$$

Предположим, что одна из телеграмм является шаблоном — т.е. имеет текст фиксированный формат, в который вписываются значения полей.

2.4 Получение второго открытого текста по первому

Допустим, что злоумышленнику этот формат известен. Тогда он получает достаточно много пар $C_1 \oplus C_2$ (известен вид обеих шифровок).

Тогда зная *P1*, имеем:

$$C_1 \oplus C_2 \oplus P_1 = P_1 \oplus P_2 \oplus P_1 = P_2$$

Таким образом, злоумышленник получает возможность определить те символы сообщения *P2*, которые находятся на позициях известного шаблона сообщения *P1*.

В соответствии с логикой сообщения *P2*, злоумышленник имеет реальный шанс узнать ещё некоторое количество символов сообщения *P2*.

Затем вновь используется описанное свойство с подстановкой вместо *P1* полученных на предыдущем шаге новых символов сообщения *P2*. И так далее.

Действуя подобным образом, злоумышленник даже если не прочитает оба сообщения, то значительно уменьшит пространство их поиска.

3 Ход работы

3.1 Исходные данные

Рассмотрим две телеграммы Центра:

- *P1* = НаВашисходящийот1204
- *P2* = ВСеверныйфилиалБанка

Ключ Центра длиной 20 байт:
K = 05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54

3.2 Код исходных данных

Установим данные значения в соответствующие поля (рис. 3.1), используя программный код, реализованный в ходе предыдущий лабораторной, и получим шифротексты.

```
Gumming g1,g2;
g1.setP("НаВашисходящийот1204");
g2.setP("ВСеверныйфилиалБанка");

int a[20] = { 0x05, 0x0C, 0x17, 0x7F, 0x0E, 0x4E, 0x37, 0xD2, 0x94, 0x10, 0x09, 0x2E, 0x22, 0x57, 0xFF, 0xC8, 0x0B, 0xB2, 0x70, 0x54 };
g1.setK(a, 20);
g2.setK(a, 20);

g1.encrypt();
g2.encrypt();
```

Figure 3.1: Исходные данные

3.3 Функция tripl()

Реализуем функцию (рис. 3.2), принимающую три строки, и возвращающую их совместное наложение операцией *XOR*, согласно описанному раннее свойству.

```
string tripl(const string& a, const string& b, const string& c) {
    //a - самая длинная строка!
    string res = a;
    for (int i = 0; i < b.length(); i++) res[i] ^= b[i];
    for (int i = 0; i < c.length(); i++) res[i] ^= c[i];
    return res;
}
```

Figure 3.2: Функция tripl()

3.4 Последовательные вызовы tripl()

Предположим, что злоумышленник знает начало первого сообщения *“НаВаш”*.

Пользуясь *tripl()* (рис. 3.3) пробуем расшифровать имеющиеся сообщения последовательной подстановкой в функцию открытых участков то первого, то второго сообщения, постепенно подбирая (рис. 3.4) продолжения уже имеющихся участков, тем самым увеличивая длину расшифрованных последовательностей.

Таким образом удалось полность расшифровать оба сообщения.

```
string s = "НаВаш";

cout << tripl(g1.getC(), g2.getC(), s) << endl;
s = "ВСеверном";
cout << tripl(g1.getC(), g2.getC(), s) << endl;
s = "ВСеверный";
cout << tripl(g1.getC(), g2.getC(), s) << endl;
s = "НаВашисходный";
cout << tripl(g1.getC(), g2.getC(), s) << endl;
s = "НаВашисходящий";
cout << tripl(g1.getC(), g2.getC(), s) << endl;
s = "ВСеверныйфилиал";
cout << tripl(g1.getC(), g2.getC(), s) << endl;
s = "НаВашисходящийответ";
cout << tripl(g1.getC(), g2.getC(), s) << endl;
s = "ВСеверныйфилиалБанка";
cout << tripl(g1.getC(), g2.getC(), s) << endl;
```

Figure 3.3: Последовательные вызовы функции *tripl()* с новыми данными

3.5 Последовательные открытия участков

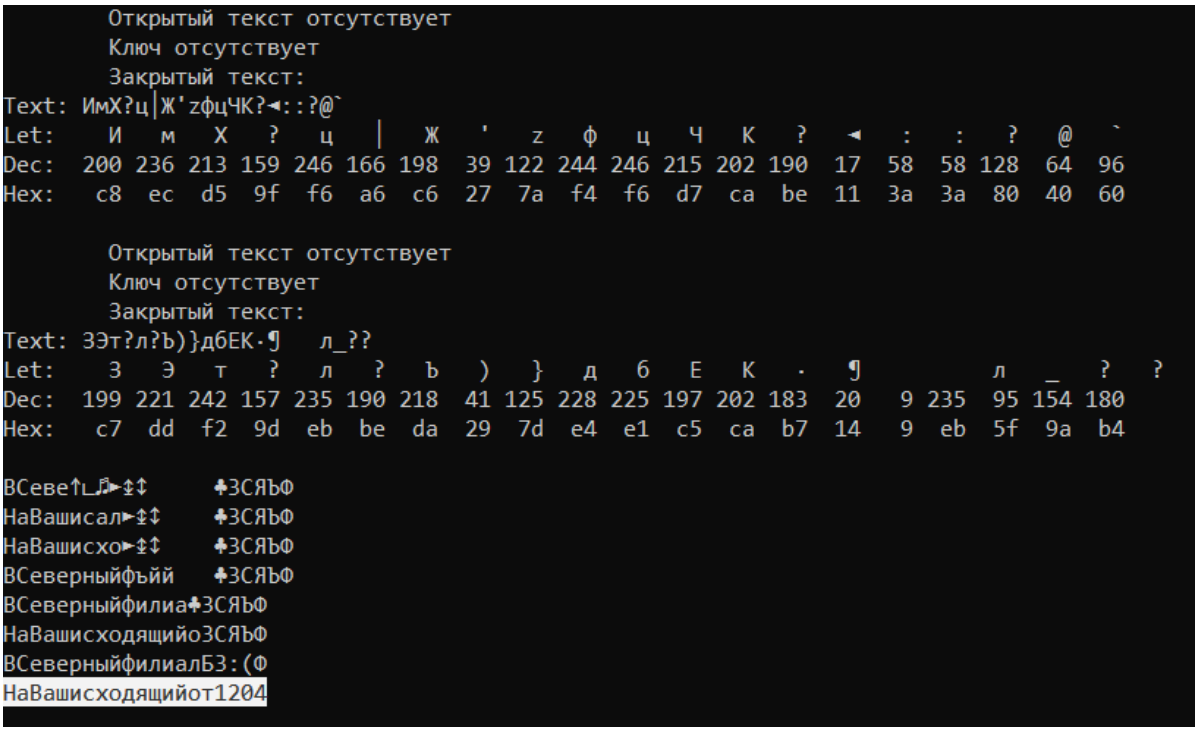

Figure 3.4: Последовательные открытия участков с новыми данными

4 Вывод

Освоили на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом.