Лабораторная работа №8

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

Ильин Андрей Владимирович

Содержание

Список иллюстраций

1 Цель работы

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

2 Задачи

Необходимо разработать приложение, позволяющее шифровать и дешифровать тексты P_1 и P_2 в режиме однократного гаммирования.Приложение должно:

1. Определить вид шифротекстов C_1 и C_2 обоих текстов P_1 и P_2 при известном ключе.

2. Требуется не зная ключа и не стремясь его определить, прочитать текст P_2, при условии, что текста подчиняются шаблону.

3 Теоретическое введение

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. [1]

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.

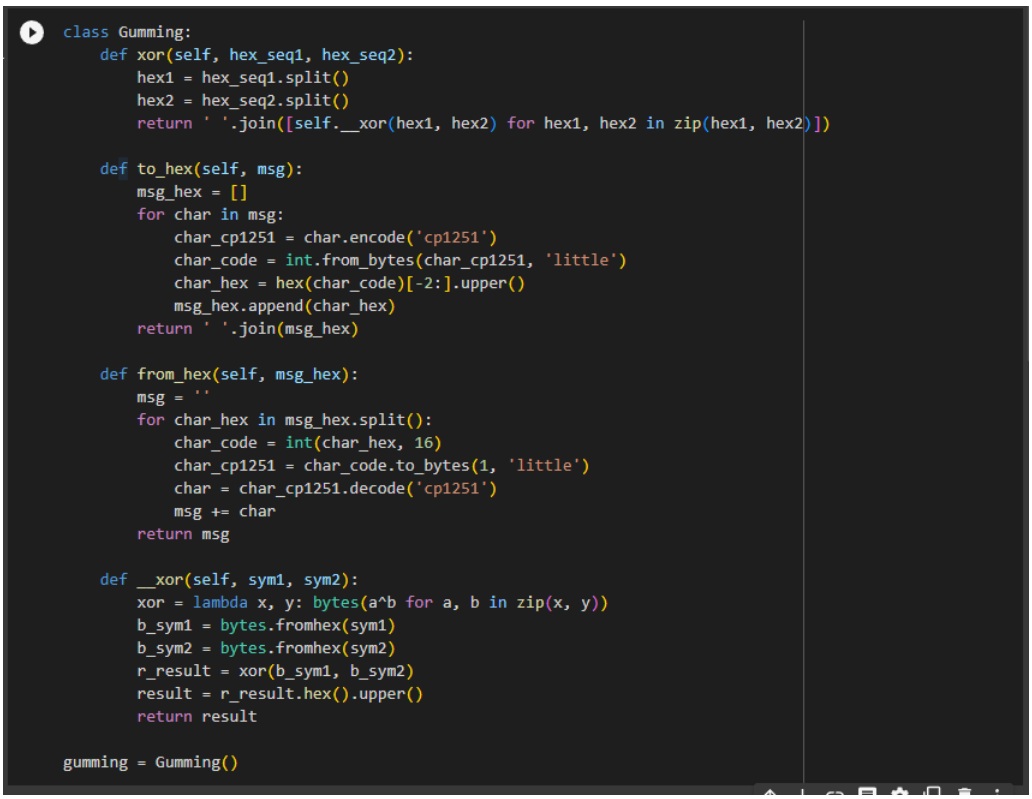
Наложение гаммы по сути представляет собой выполнение операции сложения по модулю 2 (XOR).

4 Выполнение лабораторной работы

1. Для выполнения лабораторной работы воспользуемся открытым ресурсов Google Colab. Создадим новый ноутбук - в нем будем выполнять лабораторную работу. Возьмем класс Gumming, написанный в предыдущей лабораторной работе. (рис. 4.1)

```
class Gumming:
    def xor(self, hex_seq1, hex_seq2):
        hex1 = hex_seq1.split()
        hex2 = hex_seq2.split()
        return ' '.join([self.__xor(hex1, hex2) for hex1, hex2 in zip(hex1, hex2)

    def to_hex(self, msg):
        msg_hex = []
        for char in msg:
            char_cp1251 = char.encode('cp1251')
            char_code = int.from_bytes(char_cp1251, 'little')
            char_hex = hex(char_code)[-2:].upper()
            msg_hex.append(char_hex)
        return ' '.join(msg_hex)

    def from_hex(self, msg_hex):
        msg = ''
        for char_hex in msg_hex.split():
```

```
            char_code = int(char_hex, 16)
            char_cp1251 = char_code.to_bytes(1, 'little')
            char = char_cp1251.decode('cp1251')
            msg += char
        return msg

    def __xor(self, sym1, sym2):
        xor = lambda x, y: bytes(a^b for a, b in zip(x, y))
        b_sym1 = bytes.fromhex(sym1)
        b_sym2 = bytes.fromhex(sym2)
        r_result = xor(b_sym1, b_sym2)
        result = r_result.hex().upper()
        return result

gumming = Gumming()
```

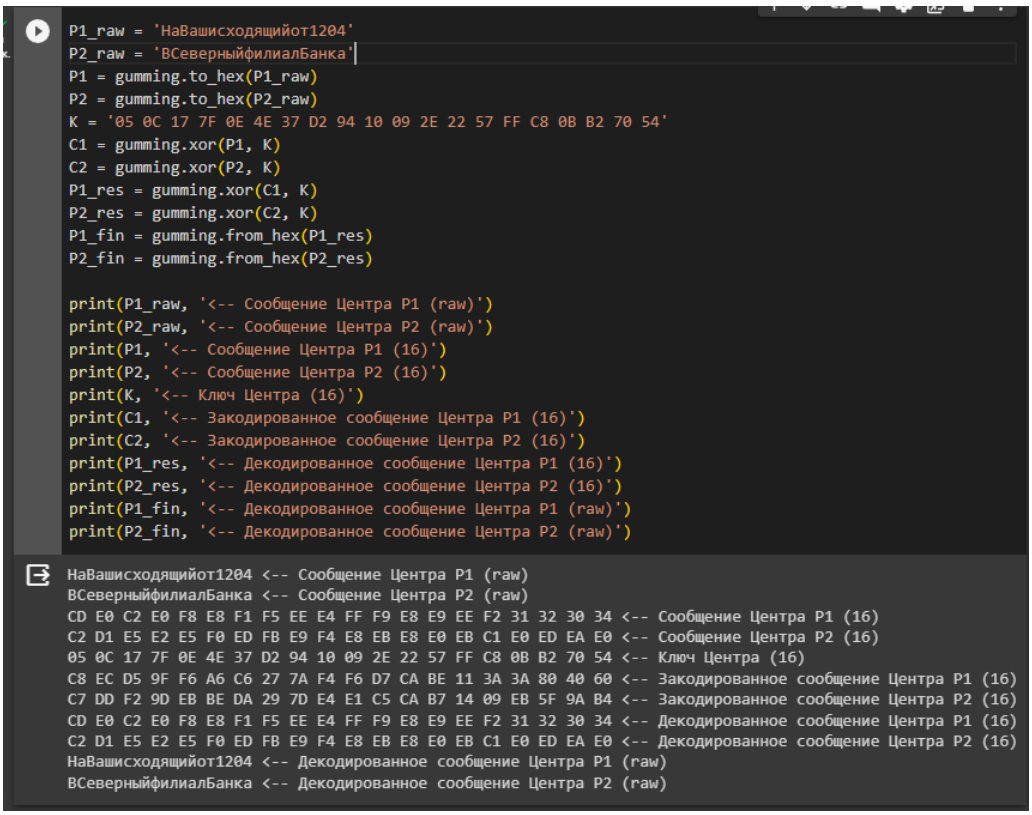


```python
class Gumming:
    def xor(self, hex_seq1, hex_seq2):
        hex1 = hex_seq1.split()
        hex2 = hex_seq2.split()
        return ' '.join([self.__xor(hex1, hex2) for hex1, hex2 in zip(hex1, hex2)])

    def to_hex(self, msg):
        msg_hex = []
        for char in msg:
            char_cp1251 = char.encode('cp1251')
            char_code = int.from_bytes(char_cp1251, 'little')
            char_hex = hex(char_code)[-2:].upper()
            msg_hex.append(char_hex)
        return ' '.join(msg_hex)

    def from_hex(self, msg_hex):
        msg = ''
        for char_hex in msg_hex.split():
            char_code = int(char_hex, 16)
            char_cp1251 = char_code.to_bytes(1, 'little')
            char = char_cp1251.decode('cp1251')
            msg += char
        return msg

    def __xor(self, sym1, sym2):
        xor = lambda x, y: bytes(a^b for a, b in zip(x, y))
        b_sym1 = bytes.fromhex(sym1)
        b_sym2 = bytes.fromhex(sym2)
        r_result = xor(b_sym1, b_sym2)
        result = r_result.hex().upper()
        return result

gumming = Gumming()
```

Рис. 4.1: Класс Gumming

2. Напишем скрипт, которой будет шифровать и дешифровать тексты P_1 и P_2 в режиме однократного гаммирования. (рис. 4.2)

```
P1_raw = 'НаВашисходящийот1204'
P2_raw = 'ВСеверныйфилиалБанка'
P1 = gumming.to_hex(P1_raw)
P2 = gumming.to_hex(P2_raw)
K = '05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54'
C1 = gumming.xor(P1, K)
C2 = gumming.xor(P2, K)
P1_res = gumming.xor(C1, K)
P2_res = gumming.xor(C2, K)
P1_fin = gumming.from_hex(P1_res)
P2_fin = gumming.from_hex(P2_res)
```



```
P1_raw = 'НаВашисходящийот1204'
P2_raw = 'ВСеверныйфилиалБанка'
P1 = gumming.to_hex(P1_raw)
P2 = gumming.to_hex(P2_raw)
K = '05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54'
C1 = gumming.xor(P1, K)
C2 = gumming.xor(P2, K)
P1_res = gumming.xor(C1, K)
P2_res = gumming.xor(C2, K)
P1_fin = gumming.from_hex(P1_res)
P2_fin = gumming.from_hex(P2_res)

print(P1_raw, '<-- Сообщение Центра P1 (raw)')
print(P2_raw, '<-- Сообщение Центра P2 (raw)')
print(P1, '<-- Сообщение Центра P1 (16)')
print(P2, '<-- Сообщение Центра P2 (16)')
print(K, '<-- Ключ Центра (16)')
print(C1, '<-- Закодированное сообщение Центра P1 (16)')
print(C2, '<-- Закодированное сообщение Центра P2 (16)')
print(P1_res, '<-- Декодированное сообщение Центра P1 (16)')
print(P2_res, '<-- Декодированное сообщение Центра P2 (16)')
print(P1_fin, '<-- Декодированное сообщение Центра P1 (raw)')
print(P2_fin, '<-- Декодированное сообщение Центра P2 (raw)')
```

```
НаВашисходящийот1204 <-- Сообщение Центра P1 (raw)
ВСеверныйфилиалБанка <-- Сообщение Центра P2 (raw)
CD E0 C2 E0 F8 E8 F1 F5 EE E4 FF F9 E8 E9 EE F2 31 32 30 34 <-- Сообщение Центра P1 (16)
C2 D1 E5 E2 E5 F0 ED FB E9 F4 E8 EB E8 E0 EB C1 E0 ED EA E0 <-- Сообщение Центра P2 (16)
05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54 <-- Ключ Центра (16)
C8 EC D5 9F F6 A6 C6 27 7A F4 F6 D7 CA BE 11 3A 3A 80 40 60 <-- Закодированное сообщение Центра P1 (16)
C7 DD F2 9D EB BE DA 29 7D E4 E1 C5 CA B7 14 09 EB 5F 9A B4 <-- Закодированное сообщение Центра P2 (16)
CD E0 C2 E0 F8 E8 F1 F5 EE E4 FF F9 E8 E9 EE F2 31 32 30 34 <-- Декодированное сообщение Центра P1 (16)
C2 D1 E5 E2 E5 F0 ED FB E9 F4 E8 EB E8 E0 EB C1 E0 ED EA E0 <-- Декодированное сообщение Центра P2 (16)
НаВашисходящийот1204 <-- Декодированное сообщение Центра P1 (raw)
ВСеверныйфилиалБанка <-- Декодированное сообщение Центра P2 (raw)
```

Рис. 4.2: Режим однократного гаммирования

3. Предположим, что каждое сообщение имеет шаблон. Сгенерируем по данному шаблону пару случайных сообщений. Зашифруем их и взломаем второе сообщение, используя шифротексты и известное первое сообщение. (рис. 4.3)

```
import random

ALPHABET = list('АаБбВвГгДдЕеЁёЖжЗзИиЙйКкЛлМмНнОоПпРрСсТтУуФфХхЦцЧчШшЩщЪъЫыЬьЭэЮю
IGNORE_SYM = '*'
P_TEMPLATE = 'На****сходящий**12**'

def getTextByTemplate(template):
    msg = ''
```

```
    for char in template:
        if char == IGNORE_SYM:
            msg += random.choice(ALPHABET)
        else:
            msg += char
    return msg

P1_raw = getTextByTemplate(P_TEMPLATE)
P2_raw = getTextByTemplate(P_TEMPLATE)

P1 = gumming.to_hex(P1_raw)
P2 = gumming.to_hex(P2_raw)
C1 = gumming.xor(P1, K)
C2 = gumming.xor(P2, K)

P2_res = gumming.xor(gumming.xor(C1, C2), P1)
P2_fin = gumming.from_hex(P2_res)
```

```python
import random

ALPHABET = list('АаБбВвГгДдЕеЁёЖжЗзИиЙйКкЛлМмНнОоПпРрСсТтУуФфХхЦцЧчШшЩщЪъЫыЬьЭэЮюЯя0123456789')
IGNORE_SYM = '*'
P_TEMPLATE = 'На****сходящий**12**'

def getTextByTemplate(template):
    msg = ''
    for char in template:
        if char == IGNORE_SYM:
            msg += random.choice(ALPHABET)
        else:
            msg += char
    return msg

P1_raw = getTextByTemplate(P_TEMPLATE)
P2_raw = getTextByTemplate(P_TEMPLATE)

P1 = gumming.to_hex(P1_raw)
P2 = gumming.to_hex(P2_raw)
C1 = gumming.xor(P1, K)
C2 = gumming.xor(P2, K)

P2_res = gumming.xor(gumming.xor(C1, C2), P1)
P2_fin = gumming.from_hex(P2_res)

print(P_TEMPLATE, '<-- Шаблон Сообщения Центра (raw)')
print(P1_raw, '<-- Сообщение Центра P1 (raw)')
print(P2_raw, '<-- Сообщение Центра P2 (raw)')
print(P1, '<-- Сообщение Центра P1 (16)')
print(P2, '<-- Сообщение Центра P2 (16)')
print(K, '<-- Ключ Центра (16)')
print(C1, '<-- Закодированное сообщение Центра P1 (16)')
print(C2, '<-- Закодированное сообщение Центра P2 (16)')
print(P1_res, '<-- Взломанное сообщение Центра P2 (16)')
print(P2_fin, '<-- Взломанное сообщение Центра P2 (raw)')
```

```
На****сходящий**12** <-- Шаблон Сообщения Центра (raw)
НаущКьсходящийуо12Ял <-- Сообщение Центра P1 (raw)
На8чОЗсходящийВд12нЁ <-- Сообщение Центра P2 (raw)
CD E0 F3 F9 CA DC F1 F5 EE E4 FF F9 E8 E9 F3 EE 31 32 DF CB <-- Сообщение Центра P1 (16)
CD E0 38 F7 CE 33 F1 F5 EE E4 FF F9 E8 E9 C2 E4 31 32 ED A8 <-- Сообщение Центра P2 (16)
05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54 <-- Ключ Центра (16)
C8 EC E4 86 C4 92 C6 27 7A F4 F6 D7 CA BE 0C 26 3A 80 AF 9F <-- Закодированное сообщение Центра P1 (16)
C8 EC 2F 88 C0 7D C6 27 7A F4 F6 D7 CA BE 3D 2C 3A 80 9D FC <-- Закодированное сообщение Центра P2 (16)
CD E0 C2 E0 F8 E8 F1 F5 EE E4 FF F9 E8 E9 EE F2 31 32 30 34 <-- Взломанное сообщение Центра P2 (16)
На8чОЗсходящийВд12нЁ <-- Взломанное сообщение Центра P2 (raw)
```

Рис. 4.3: Взлом второго текста

5 Анализ результатов

Работа выполненна без непредвиденных проблем в соответствии с руководством. Ошибок и сбоев не произошло.

6 Выводы

Нам удалось освоить на практике применение режима однократного гаммирования, в дополнение закрпеили навки владения языками программирования, в частности языком программирования - `python`.

Список литературы

1. Материалы по лабораторной работе [Электронный ресурс]. RUDN. URL: https://esystem.rudn.ru/pluginfile.php/2090286/mod_resource/content/2/008-lab_crypto-key.pdf.