

Отчёт по лабораторной работе

Лаб 7

Аристид Жан Лоэнс Аристобуль

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	11

Список иллюстраций

3.1	шестнадцатеричное представление	7
3.2	Random Key	7
3.3	Двойчное представление	8
3.4	Зашифрованный текст	9
3.5	Дешифрованный текст в двойчном представлении	9
3.6	дешифрованный текст	10
3.7	Новый Ключ	10
3.8	Неправильное сообщение	10

Список таблиц

1 Цель работы

Освоить на практике применение режима однократного гаммирования¹

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 Выполнение лабораторной работы

Здесь мы представляем открытый текст на шестнадцатеричном представлении (рис. 3.1).



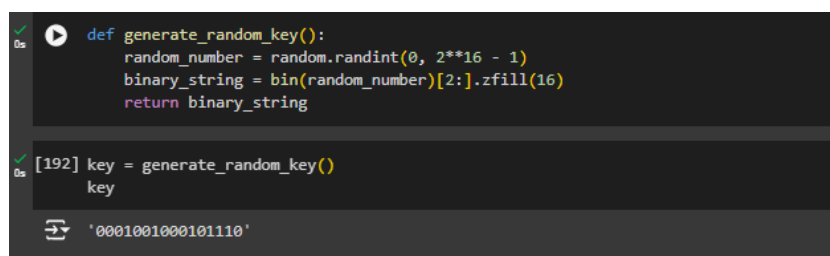
```
[177] import random

[190] text = "С Новым Годом, друзья!"
      utf16_bytes = text.encode("utf-16")
      print(utf16_bytes)

b'\xff\xfe!\x04 \x00\x1d\x04>\x042\x04K\x04<\x04 \x00\x13\x04>\x044\x04>\x04<\x04,\x04
```

Рис. 3.1: шестнадцатеричное представление

Здесь генерируемый ключ шифрования случайным образом (рис. 3.2).



```
def generate_random_key():
    random_number = random.randint(0, 2**16 - 1)
    binary_string = bin(random_number)[2:].zfill(16)
    return binary_string

[192] key = generate_random_key()
      key

'0001001000101110'
```

Рис. 3.2: Random Key

Здесь мы представляем открытый текст в двоичном представлении (рис. 3.3).

```
[193] def utf16_to_binary_strings(utf16_bytes):
    if len(utf16_bytes) % 2 != 0:
        raise ValueError("UTF-16 byte sequence must have an even length.")

    binary_strings = []
    for i in range(0, len(utf16_bytes), 2):
        code_point = (utf16_bytes[i+1] << 8) | utf16_bytes[i]
        binary_string = bin(code_point)[2:].zfill(16) # Convert to binary string,
        binary_strings.append(binary_string)
    return binary_strings

binary_strings = utf16_to_binary_strings(utf16_bytes)

[195] binary_strings

['1111111011111111',
'0000010000100001',
'0000000000100000',
'0000010000011101',
'0000010000111110',
'0000010000110010',
'0000010001001011',
'0000010000111100',
'0000000000100000',
'0000010000010011',
'0000010000111110',
'0000010000110100',
'0000010000111110',
'0000010000111100',
'0000000000101100',
'0000000000100000']
```

Рис. 3.3: Двойчное представление

Функция `xor_16bit_strings()` реализует операция сложение по модулю 2 чтобы шифровать текст. (рис. 3.4).


```

[196] def xor_16bit_strings(string_a, key):
    """Performs XOR on two 16-bit binary strings."""

    # Ensure equal length
    if len(string_a) != 16 or len(key) != 16:
        raise ValueError("Strings must be 16 bits long.")

    result = ""
    for i in range(16):
        result += str(int(string_a[i]) ^ int(key[i]))
    return result

[197] encrypted_binary = []
      for b in binary_strings:
          encrypted_binary.append(xor_16bit_strings(b, key))

[198] encrypted_binary

```

```

['1110110011010001',
 '0001011000001111',
 '0001001000001110',
 '0001011000110011',
 '0001011000010000',
 '0001011000011100',
 '0001011001100101',
 '0001011000010010',
 '0001001000001110',
 '0001011000111101',
 '0001011000010000',
 '0001011000011010',
 '0001011000010000',

```

Рис. 3.4: Зашифрованный текст

Здесь у нас дешифрованный текст в двоичном представлении. (рис. 3.5).

```

[212] decrypted_binary = []
      for e in encrypted_binary:
          decrypted_binary.append(xor_16bit_strings(e, key))

decrypted_binary

```

```

['1111110111111111',
 '0000010000100001',
 '0000000000100000',
 '0000010000011101',
 '0000010000111110',
 '0000010000110010',
 '0000010001001011',
 '0000010000111100',
 '0000000000100000',
 '0000010000010011',
 '0000010000111110',
 '0000010000110100',
 '0000010000111110',
 '0000010000111100',
 '0000000000101100',
 '0000000000100000',
 '0000010000110100',
 '0000010001000000',
 '0000010001000011',
 '0000010000110111',
 '0000010001001100',
 '0000010001001111',
 '0000000000100001']

```

Рис. 3.5: Дешифрованный текст в двоичном представлении

Здесь Мы получили сообщение после дешифрования (рис. 3.6).

4 Выводы

В ходе этой лабораторной работы мы изучили хороший метод криптографии для отправки сообщений, которые могут быть поняты только теми, у кого есть ключ дешифрования.