

Отчёт по лабораторной работе

Лаб 8

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	8
3.5	Дешифрованный текст в двойном представлении	9
3.6	дешифрованный текст	9
3.7	Новый Ключ	10

Список таблиц

1 Цель работы

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

2 Задание

Два текста кодируются одним ключом (однократное гаммирование). Требуется не зная ключа и не стремясь его определить, прочесть оба текста. Необходимо разработать приложение, позволяющее шифровать и дешифровать тексты P1 и P2 в режиме однократного гаммирования. Приложение должно определить вид шифротекстов C1 и C2 обоих текстов P1 и P2 при известном ключе ; Необходимо определить и выразить аналитически способ, при котором злоумышленник может прочесть оба текста, не зная ключа и не стремясь его определить

3 Выполнение лабораторной работы

Здесь мы представляем открытые тексты P1 и P2 на шестнадцатеричном представлении (рис. 3.1).



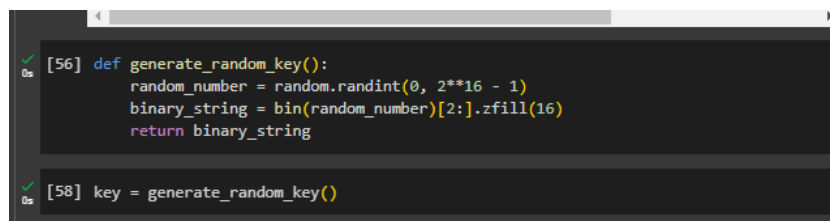
```
[8] import random

[55] P1 = "НаВашисходящийот1204"
      P2 = "ВСеверныйфилиалБанка"
      utf16_bytes1 = P1.encode("utf-16")
      utf16_bytes2 = P2.encode("utf-16")
      print(utf16_bytes1)
      print(utf16_bytes2)

b'\xff\xfe\x1d\x040\x04\x12\x040\x04H\x048\x04A\x04E\x04>\x044\x040\x04I\x048\x049\x04
b'\xff\xfe\x12\x04!\x045\x042\x045\x04@\x04=\x04K\x049\x04D\x048\x04;\x048\x040\x04
```

Рис. 3.1: шестнадцатеричное представление

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



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

[58] key = generate_random_key()
```

Рис. 3.2: Random Key

Здесь мы представляем открытые тексты в двоичном представлении (рис. 3.3).

```

[59] 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

[60] P1_b = utf16_to_binary_strings(utf16_bytes1)
    P2_b = utf16_to_binary_strings(utf16_bytes2)

P2_b

```

```

['1111111101111111',
 '0000010000010010',
 '0000010000100001',
 '0000010000110101',
 '0000010000110010',
 '0000010000110101',
 '0000010001000000',
 '0000010000111101',
 '0000010001001011',
 '0000010000111001',
 '0000010001000100',
 '0000010000111000',
 '0000010000111011',
 '0000010000111000',
 '0000010000110000']

```

Рис. 3.3: Двойчное представление

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

```

def xor_16bit_strings(string_a, key):
    # Ensure equal length
    if len(string_a) != len(key):
        raise ValueError("Strings must be 16 bits long.")

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

[63] C1 = []
    C2 = []
    for b in P1_b:
        C1.append(xor_16bit_strings(b, key))
    for b in P2_b:
        C2.append(xor_16bit_strings(b, key))

```

Рис. 3.4: Зашифрованный текст

Здесь у нас дешифрованные тексты в двойчном представлении. (рис. 3.5).


```

}  ▾ DECRYPTION

[64] decrypted_binary1 = []
    decrypted_binary2 = []
    for e in C1:
        decrypted_binary1.append(xor_16bit_strings(e, key))
    for e in C2:
        decrypted_binary2.append(xor_16bit_strings(e, key))

0s  ▶ decrypted_binary1

[ '1111111011111111',
  '0000010000011101',
  '0000010000110000',
  '0000010000010010',
  '0000010000110000',
  '0000010001001000',
  '0000010000111000',
  '0000010001000001',
  '0000010001000101',
  '0000010000111110',
  '0000010000110100',
  '0000010001001111',
  '0000010001001001',
  '0000010000111000',
  '0000010000111001',
  '0000010000111110' ]

```

Рис. 3.5: Дешифрованный текст в двоичном представлении

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

```

[66] def binary_to_text(decrypted_binary):
    text = ""
    for i in range(1, len(decrypted_binary)):
        code_point = decrypted_binary[i]
        text += chr(int(code_point, 2))
    return text

    decrypted_text1 = binary_to_text(decrypted_binary1)
    decrypted_text2 = binary_to_text(decrypted_binary2)
    print(decrypted_text1)
    print(decrypted_text2)

НаВашисходящийот1204
ВСеверныйфилиалБанка

```

Рис. 3.6: дешифрованный текст

Мы используем другой метод без знания ключа что дешифровать. (рис. 3.7).

```
✓ [53] res_xor=[]  
0s   for i in range(0, len(C1)):  
       res_xor.append(xor_16bit_strings(C1[i], C2[i]))  
       for j in range(0, len(C1)):  
           res_xor[j] = xor_16bit_strings(res_xor[j], P1_b[j])  
  
✓ [54] raw_text = binary_to_text(res_xor)  
0s   raw_text  
  
↔ "ВСеверныйфилиалБанка"
```

Рис. 3.7: Новый Ключ

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.