Лабораторная работа №7

Элементы криптографии. Однократное гаммирование

Роман Владимирович Иванов

Содержание

1 Цель работы **5**

2 Задание **6**

3 Выполнение лабораторной работы **7**

4 Выводы **9**

5 Ответы на контрольные вопросы **10**

6 Список литературы **12**

Список иллюстраций

3.1 Функция, шифрующая данные 7
3.2 Результат работы функции, шифрующей данные 7
3.3 Функция, дешифрующая данные 8
3.4 Результат работы функции, дешифрующей данные 8
3.5 Сравнение ключей 8

Список таблиц

1 Цель работы

Освоить на практике применение режима однократного гаммирования [1].

2 Задание

1. Написать программу, которая должна определить вид шифротекста при известном ключе и известном открытом тексте
2. Также эта программа должна 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

1. Написал функцию шифрования, которая определяет вид шифротекста при известном ключе и известном открытом тексте “С Новы Годом, друзья!”. Ниже представлены функция, шифрующая данные (рис - @fig:001), а также работа данной функции (рис - @fig:002).

Ввод [1]:
```
import numpy as np
```

Ввод [2]:
```
def encryption(text):
    print("Открытый текст: ", text)
    # Задам массив из символов открытого текста в шестнадцатеричном представлении:
    text_array = []
    for i in text:
        text_array.append(i.encode("cp1251").hex())
    print("\nОткрый текст в шестнадцатеричном представлении: ", *text_array)

    # Задам случайно сгенерированный ключ в шестнадцатеричном представлении:
    key_dec = np.random.randint(0, 255, len(text))
    key_hex = [hex(i)[2:] for i in key_dec]
    print("\nКлюч в шестнадцатеричном представлении: ", *key_hex)

    # Задам зашифрованный текст в шетснадцатеричном представлении:
    crypt_text = []
    for i in range(len(text_array)):
        crypt_text.append("{:02x}".format(int(text_array[i], 16) ^ int(key_hex[i], 16)))
    print("\nЗашифрованный текст в шестндцатеричном представлении: ", *crypt_text)

    # Задам зашифрованный текст в обычном представлении:
    final_text = bytearray.fromhex("".join(crypt_text)).decode("cp1251")
    print("\nЗашифрованный текст: ", final_text)
    return key_hex, final_text
```

Рис. 3.1: Функция, шифрующая данные

Ввод [4]:
```
# Изначальная фраза:
phrase = "С Новым Годом, друзья!"
# Получение сгенерированного ключа и зашифрованной фразы:
crypt_key, crypt_text = encryption(phrase)
```

```
Открытый текст:  С Новым Годом, друзья!

Откры текст в шестнадцатеричном представлении:  d1 20 cd ee e2 fb ec 20 c3 ee e4 ee ec 2c 20 e4 f0 f3 e7 fc ff 21

Ключ в шестнадцатеричном представлении:  bd 34 23 ce 18 10 89 50 12 92 62 c1 83 78 ac 54 2b 17 36 9b ef 1

Зашифрованный текст в шестндцатеричном представлении:  6c 14 ee 20 fa eb 65 70 d1 7c 86 2f 6f 54 8c b0 db e4 d1 67 10 20

Зашифрованный текст:  l□о ълерС|Ў/оТЊ°ЫдС□g□
```

Рис. 3.2: Результат работы функции, шифрующей данные

2. Написал функцию дешифровки, которая определяет ключ, с помощью которого шифротекст может быть преобразован в некоторый фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста (рис - @fig:003). А также представил результаты работы программы (рис - @fig:004).

Ввод [3]:
```
def decryption(text, final_text):
    print("Открытый текст: ", text)
    print("\nЗашифрованный текст: ", final_text)

    # Задам массив из символов открытого текста в шестнадцатеричном представлении:
    text_hex = []
    for i in text:
        text_hex.append(i.encode("cp1251").hex())
    print("\nОткрый текст в шестнадцатеричном представлении: ", *text_hex)

    # Задам массив из символов зашифрованного текста в шестнадцатеричном представлении:
    final_text_hex = []
    for i in final_text:
        final_text_hex.append(i.encode("cp1251").hex())
    print("\nЗашифрованный текст в шестнадцатеричном представлении: ", *final_text_hex)

    # Найду ключ:
    key = [hex(int(i, 16) ^ int(j, 16))[2:] for (i, j) in zip(text_hex, final_text_hex)]
    print("\nНужный ключ в шестнадцатеричном представлении: ", *key)
    return key
```

Рис. 3.3: Функция, дешифрующая данные

Ввод [5]:
```
# Получение нужного ключа:
key = decryption(phrase, crypt_text)
```
```
Открытый текст:  С Новым Годом, друзья!

Зашифрованный текст:  l□о ълерС|†/оТЊ°ЫдЄg□

Открый текст в шестнадцатеричном представлении:  d1 20 cd ee e2 fb ec 20 c3 ee e4 ee ec 2c 20 e4 f0 f3 e7 fc ff 21

Зашифрованный текст в шестнадцатеричном представлении:  6c 14 ee 20 fa eb 65 70 d1 7c 86 2f 6f 54 8c b0 db e4 d1 67 10 20

Нужный ключ в шестнадцатеричном представлении:  bd 34 23 ce 18 10 89 50 12 92 62 c1 83 78 ac 54 2b 17 36 9b ef 1
```

Рис. 3.4: Результат работы функции, дешифрующей данные

Сравнение ключей, полученных с помощью первой и второй функций (рис - @fig:005).

Ввод [6]:
```
# Проверка правильности ключа:
print("Ключ верен!") if crypt_key == key else print("Ключ неверен!")
```
```
Ключ верен!
```

Рис. 3.5: Сравнение ключей

4 Выводы

Освоил на практике применение режима однократного гаммирования.

5 Ответы на контрольные вопросы

1. Одократное гаммирование - выполнение операции XOR между элементами гаммы и элементами подлежащего сокрытию текста. 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.

2. Недостатки однократного гаммирования: Абсолютная стойкость шифра доказана только для случая, когда однократно используемый ключ, длиной, равной длине исходного сообщения, является фрагментом истинно случайной двоичной последовательности с равномерным законом распределения.

3. Преимущества однократного гаммирования: во-первых, такой способ симметричен, т.е. двойное прибавление одной и той же величины по модулю 2 восстанавливает исходное значение; во-вторых, шифрование и расшифрование может быть выполнено одной и той же программой. Наконец, 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.

4. Длина открытого текста должна совпадать с длиной ключа, т.к. если ключ

короче текста, то операция XOR будет применена не ко всем элементам и конец сообщения будет не закодирован, а если ключ будет длиннее, то появится неоднозначность декодирования.

5. Операция XOR используется в режиме однократного гаммирования. Наложение гаммы по сути представляет собой выполнение побитовой операции сложения по модулю 2, т.е. мы должны сложить каждый элемент гаммы с соответствующим элементом ключа. Данная операция является симметричной, так как прибавление одной и той же величины по модулю 2 восстанавливает исходное значение.
6. Получение шифротекста по открытому тексту и ключу: $C_i = P_i \oplus K_i$

7. Получение ключа по окрытому тексту и шифротексту: $K_i = P_i \oplus C_i$

8. Необходимы и достаточные условия абсолютной стойкости шифра: полная случайность ключа; равенство длин ключа и открытого текста; однократное использование ключа.

6 Список литературы

1. Кулябов Д. С., Королькова А. В., Геворкян М. Н. Информационная безопасность компьютерных сетей. Лабораторная работа № 7. Элементы криптографии. Однократное гаммирование.