Отчёт по лабораторной работе

Лабораторная №8

Панкратьев Александр Владимирович

Содержание

List of Tables

List of Figures

1 Цель работы

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

2 Теоретическая часть

Если даны две телеграммы Центра, то шифротексты обеих телеграмм можно получить по формулам режима однократного гаммирования: C1 = P1 ^ K, C2 = P2 ^ K. Чтобы найти открытый текст, зная шифротекст двух телеграмм, зашифрованных одним ключом, надо сложить по модулю 2 эти два равенства. Тогда с учётом свойства операции XOR

1 ^ 1 = 0, 1 ^ 0 = 1 ,

получаем:

C1 ^ C2 = P1 ^ K ^ P2 ^ K = P1 ^ P2.

Предположим, что одна из телеграмм является шаблоном — т.е. имеет текст фиксированный формат, в который вписываются значения полей. Допустим, что злоумышленнику этот формат известен. Тогда он получает достаточно много пар C1 ^ C2 (известен вид обеих шифровок). Тогда зная P1, имеем:

C1 ^ C2 ^ P1 = P1 ^ P2 ^ P1 = P2.

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

3 Выполнение лабораторной работы

Написал программу на языке Python, позволяющую шифровать и дешифровать данные в режиме однократного гаммирования. Программа имеет 3 функции:

1. decode(cr_message, key). Данная функция принимает зашифрованное сообщение и ключ (в виде строк с шестнадцатиричными значениями). Для каждого значения зашифрованного сообщения выполняется сложение по модулю 2 с сответствующим значением ключа. Функция возвращает строку с расшифрованным сообщением (рис. 3.1).

```
2  def decode(cr_message, key):
3      message = []
4      cr_message = cr_message.split()
5      key = key.split()
6      for i in range(0, len(cr_message)):
7          message.append(chr(int(cr_message[i], 16) ^ int(key[i], 16)))
8      return ''.join(message)
```

Figure 3.1: Функция для дешифрования сообщения

2. def encode(message, key). Данная функция принимает исходное сообщение и ключ. Каждый символ сообщения преобразовывется в число, соответствующее его коду в системе Unicode. Далее выполняется сложение по модулю 2 между получившимися кодами и соответствующими значениями ключа. Функция возвращает зашифрованное сообщение в виде строки с шестнадцатиричными значениями (рис. 3.2).

```python
def encode(message, key):
    cr_message = []
    key = key.split()
    for i in range(0, len(message)):
        cr_message.append((hex(ord(message[i]) ^ int(key[i], 16)).lstrip('0x')).upper())
        if len(cr_message[i]) == 1:
            cr_message[i] = '0' + cr_message[i]
    return ' '.join(cr_message)
```

Figure 3.2: Функция для шифрования сообщения

3. get_message(cr_message1, cr_message2, message2). Данная функция принимает зашифрованное сообщение, шаблон исходного сообщения и зашифрованное шаблонное сообщение. Выполняется сложение по модулю 2 между значениями закодированного сообщения, кодами символов шаблонного сообщения и значениями закодированного шаблонного сообщения. Функция возвращает строку с расшифрованным сообщением, с помощью которого исходный текст был закодирован (рис. 3.3).

```python
def get_message(cr_message1, cr_message2, message2):
    message1 = []
    cr_message1 = cr_message1.split()
    cr_message2 = cr_message2.split()
    for i in range(0, len(cr_message1)):
        message1.append(chr(int(cr_message1[i], 16) ^ int(cr_message2[i], 16) ^ ord(message2[i])))
    return ''.join(message1)
```

Figure 3.3: Функция для дешифрования сообщения без ключа

Написал код с вызовом функций для тестирования (рис. 3.4).

```python
print('Определим вид шифротекста_1 при известном ключе и известном открытом тексте')
message = input('Введите текст сообщения_1: ')
key = input('Введите ключ: ')
cr_message_test = encode(message, key)
print('Закодированное сообщение_1:', cr_message_test)

print()
print('Определим вид шифротекста_2 при известном ключе и известном открытом тексте')
message = input('Введите текст сообщения_2: ')
key = input('Введите ключ: ')
cr_message_test = encode(message, key)
print('Закодированное сообщение_2:', cr_message_test)

print()
print('Декодируем сообщение при его известном шифротексте, имея известный шаблон и его шифротекст')
cr_message1 = input('Введите текст закодированного сообщения: ')
message2 = input('Введите текст шаблонного сообщения: ')
cr_message2 = input('Введите текст закодированного шаблонного сообщения: ')
message1 = get_message(cr_message1, cr_message2, message2)
print('Декодированное сообщение:', message1)
```

Figure 3.4: Вызов функций для тестирования

Протестировал программу на сообщениях ‘С Новым Годом, друзья!’ и ‘Желаю счастья и любви!’. Вначале программа определила вид шифротекста сообщений (при одинаковом

ключе). Далее была вызвана функция get_message(), в которую были переданы шифротекст первого сообщения, а также исходный текст и шифротекст второго сообшения. После обработки этих данных, функция корректно определила исходный текст первого сообщения (рис. 3.5).

Figure 3.5: Тестирование программы

4 Вывод

Я освоил на практике применение режима однократного гаммированияна примере кодирования различных исходных текстов одним ключом. Определила способ, при котором злоумышленник может прочитать оба текста, не зная ключа.