

Лабораторная работа №7

Основы информационной безопасности

Феоктистов Владислав Сергеевич

22 сентября 2022

Российский университет дружбы народов, Москва, Россия

НПМбд-01-19

Элементы криптографии.
Однократное гаммирование.

Целью данной работы является освоение на практике применение режима однократного гаммирования.

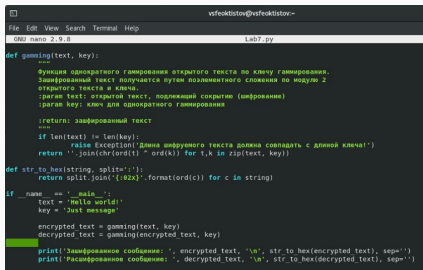
- Написать программу, позволяющую шифровать и дешифровать данные в режиме однократного гаммирования;
- Определить вид шифротекста при известном ключе и известном открытом тексте;
- Определить ключ, с помощью которого шифротекст может быть преобразован в некоторый фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста

Гаммирование представляет собой наложение (снятие) на открытые (зашифрованные) данные последовательности элементов других данных (ключа), чаще всего того же размера (ключ можно зациклить). Под наложением, по сути, подразумевается выполнение операции сложения по модулю 2 (XOR) (обозначаемая знаком \oplus) между элементами гаммы (ключа) и элементами, подлежащих сокрытию. Такой метод шифрования является симметричным, так как двойное прибавление одной и той же величины по модулю 2 восстанавливает исходное значение, а шифрование и расшифрование выполняется одной и той же программой (функцией).

Ход выполнения лабораторной работы

Функция однократного гаммирования и представления строки

Написали функцию, производящую однократное гаммирование посредством побитового сложения по модулю 2 элементов открытого текста и ключа, а так же функцию представления строки в виде последовательности символов в шестнадцатиричном виде, разделенных двоеточиями.



```
vsfeeklistov@vsfeeklistov:~$ nano Lab7.py
GNU nano 2.9.8 Lab7.py

def gaming(text, key):
    """
    Функция однократного гаммирования открытого текста по ключу гаммирования.
    Зашифрованный текст получается путем поэлементного сложения по модулю 2
    открытого текста и ключа.
    :param text: открытый текст, подлежащий сокрытию (шифрование)
    :param key: ключ для однократного гаммирования
    :return: зашифрованный текст
    """
    if len(text) != len(key):
        raise Exception('Длина шифруемого текста должна совпадать с длиной ключа!')
    return ''.join(chr(ord(t) ^ ord(k)) for t,k in zip(text, key))

def str_to_hex(string, split=''):
    return split.join('{:02x}'.format(ord(c)) for c in string)

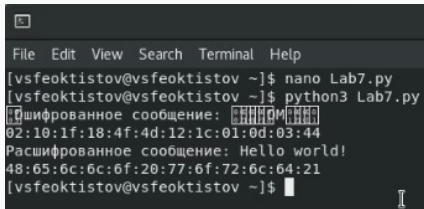
if __name__ == '__main__':
    text = 'Hello world!'
    key = 'Just message'

    encrypted_text = gaming(text, key)
    decrypted_text = gaming(encrypted_text, key)

    print('Зашифрованное сообщение: ', encrypted_text, '\n', str_to_hex(encrypted_text), sep='')
    print('Расшифрованное сообщение: ', decrypted_text, '\n', str_to_hex(decrypted_text), sep='')
```

Figure 1: Код программы. Первая версия

Однократное гаммирование открытого текста приводит к его шифрованию, а однократное гаммирование зашифрованного текста тем же ключом - к расшифровке.



```
File Edit View Search Terminal Help
[vsfeoktistov@vsfeoktistov ~]$ nano Lab7.py
[vsfeoktistov@vsfeoktistov ~]$ python3 Lab7.py
Шифрованное сообщение: 02:10:1f:18:4f:4d:12:1c:01:0d:03:44
02:10:1f:18:4f:4d:12:1c:01:0d:03:44
Расшифрованное сообщение: Hello world!
48:65:6c:6c:6f:20:77:6f:72:6c:64:21
[vsfeoktistov@vsfeoktistov ~]$
```

Figure 2: Запуск программы. Первая версия

Функция генерации необходимого ключа

Написали функцию генерации ключа для указанного текста, которое будет после однократного гаммирования этого текста этим ключом выдавать необходимый текст. По факту эта функция является оберткой функции *gamming*.

```
def get_key(text, required_text):
    """
    Функция получения ключа, данного plaintext, после
    однократного гаммирования ciphertext.
    :param text: открытый текст, который в дальнейшем нужно
    будет гаммировать однократным гаммированием выданным ключом
    :param required_text: желаемый текст, который нужно получить
    после однократного гаммирования указанного сообщения выданным ключом
    """
    :return: желаемый ключ гаммирования
    """
    if len(text) != len(required_text):
        raise Exception('Длина шифруемого текста должна совпадать с длиной plaintext после однократного гаммирования!')
    return gamming(text, required_text)
```

Figure 3: Функция генерации желаемого ключа

Функция генерации случайной строки

Для удобства написали функцию генерации случайной строки заданной длины *length* с использованием заданных символов *letters*, причем по умолчанию - это латинские символы верхнего и нижнего регистров.

```
def generate_random_str(length, letters=''):
    """
    Функция возвращает строку со случайной последовательностью символов,
    которые указаны в переменной letters. По умолчанию - латинские символы
    верхнего и нижнего регистра.
    :param length: длина желаемой строки
    :param letters: символы, которые будут использоваться для создания строки.
    По умолчанию - латинские символы верхнего и нижнего регистра.
    :return: строка последовательности случайных символов, указанных в параметре
    letters.
    """
    if letters == '':
        letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    return ''.join(letters[randint(0, len(letters)-1)] for i in range(length))
```

Figure 4: Функция генерации случайной строки

Код проверки работы всех функций

Написали небольшой код для проверки работы всех функций на примерах. Открытый текст будет содержать сообщение “С Днем народного единства!”, а ключ гаммирования сгенерируется случайным образом. В результате смодем получить зашифрованный текст после первого гаммирования и расшифрованный - после второго. Также добавили переменную с желаемым текстом, получаемого после однократного гаммирования указанного сообщения, - “С Новым годом, друзья!”.

```
if __name__ == '__main__':
    text = 'С Днем народного единства!' # len = 26
    key = generate_random_str(len(text)) # len = 26

    print('Исходный открытый текст: ', text, '\n', str_to_hex(text), sep='')
    print('Ключ гаммирования: ', key, '\n', str_to_hex(key), sep='')

    encrypted_text = gammimg(text, key)
    decrypted_text = gammimg(encrypted_text, key)

    print('Зашифрованное сообщение: ', encrypted_text, '\n', str_to_hex(encrypted_text), sep='')
    print('Расшифрованное сообщение: ', decrypted_text, '\n', str_to_hex(decrypted_text), sep='')

    required_text = 'С Новым Годом, друзья!' # len = 26
    required_key = get_key(encrypted_text, required_text)
    decrypted_required_text = gammimg(encrypted_text, required_key)

    print('Желаемый ключ: ', required_key, '\n', str_to_hex(required_key), sep='')
    print('Расшифровка сообщения ключом, которое дает желаемое сообщение для известного шифротекста: ',
          decrypted_required_text, '\n', str_to_hex(decrypted_required_text))
```

Figure 5: Код проверки работы всех функций

Запуск программы. Вторая версия

Для зашифрованного сообщения можно подобрать ключ, который после применения однократного гаммирования даст желаемое сообщение, отличное от начального (сообщения до шифрования).

[illegible]

Figure 6: Запуск программы. Вторая версия

В процессе выполнения лабораторной работы освоил на практике применение режима однократного гаммирования и написал программу, реализующую это шифрование.