Лабораторная Работа №9.

Основы информационной безопасности

Дудырев Г.А.

Российский университет дружбы народов им. Патриса Лумумбы, Москва, Россия

Докладчик

- Дудырев Глеб Андреевич
- НПИбд-01-22
- Российский университет дружбы народов
- [1132222003@pfur.ru]
- https://github.com/GlebDudyrev

Цель работы

Цель работы

Освоить на практике применение режима однократного гаммирования1

Выполнение лабораторной работы

Функция generate_key

Для начала реализую функцию generate_key, которая генерирует случайнную последовательность бит, то есть секретный ключ.

```
def generate_key(key_len: int) -> list:
    """
        Функция генерирует пвсевдослучайную последовательность,
        которая будет использоваться в качестве ключа для шифрования.
    """
    key = [] #Объект, который будет содержать итоговую последовательность
    for i in range(key_len):
        #Генерируем последовательность
        key.append(np.random.randint(0, 2))
    return key
```

Функция generate_key

Посмотрим на результат работы generate_key():

```
#Examples
#Сгенерируем секретный ключ длиной 10
key = ''.join([str(x) for x in generate_key(10)])
print('Сгенерированный ключ: {}'.format(key))
```

✓ 0.0s

```
Сгенерированный ключ: 1011001101
```

Figure 1: Функция generate_key()

Функция encrypt

Далее я реализую функцию encrypt, которая производит шифрование открытого текста, с помощью применения однократного кодирования.

```
def encrypt(open_text: str, key: list = None) -> str:
    """
        Функция шифрует данные в режиме однократного гаммирования.
    """
    #Из открытого текста получаем бинарную последовательность
    open_text_bin = ''.join(format(ord(x), '08b') for x in open_text)
    #Если ключ не передается, то сгенерируем его
    if not key:
        key_len = len(open_text_bin)
        key = generate_key(key_len)
    #Получаем последовательность бит шифротекста, применяя последовательно XOR к биту из
    ciphertext_bin = []
```

Посмотрим на пример работы этой функции:

```python
#Examples
open_text = 'Happy New Year, my friends!'
print(f'Открытый текст: {open_text}')
cipher_text, key = encrypt(open_text)
print('Серкетный ключ: {}'.format(''.join([str(x) for x in key])))
print(f'Шифротекст: {cipher_text}')
```

✓ 0.0s — Python

```
Открытый текст: Happy New Year, my friends!
Серкетный ключ: 101100111110010010010100111111011001110101101101010001100100011010101110111000000001100100000100100001001111110100110011110000010111011000010110110
Шифротекст: û▯ä▯ä#ÙÀ@aå▯▯á▯oáLù▯L"Ü~]
```

Figure 2: Функция encrypt()

Функция decrypt

Следующим шагом я реализовал функцию, которая призводит дешифрования шифротекста.

```
def decrypt(cipher_text: str, key) -> str:
    """
        Функция, которая производит дешифрование
    """
    if not key: #Если ключ не передали, то завершаем работу программы
        return 'You should enter the secret key.'
    #Из зашифрованного текста получаем бинарную последовательность
    cipher_text_bin = ''.join(format(ord(x), '08b') for x in cipher_text)
    #Получаем последовательность бит открытого текста, применяя последовательно XOR к би
    open_text_bin = []
    for idx, bit in enumerate(cipher_text_bin):
        open_text_bin.append(int(bit) ^ key[idx])
```

Функция decrypt

Посмотрим на пример ее работы:

```
#Examples
print(f'Шифротекст: {cipher_text}')
res_text, key = decrypt(cipher_text, key)
print('Серкетный ключ: {}'.format(''.join([str(x) for x in key])))
print(f'Текст после дешифровки: {res_text}')
```

✓ 0.0s

```
Шифротекст: û□ä□ä#ÙÀ@aå□□á□oáLù□L"Ü~]
Серкетный ключ: 10110011111001001001010011111101100111010110110101000110010001101010111011100
Текст после дешифровки: Happy New Year, my friends!
```

Figure 3: Функция decrypt()

Функция key_find

Далее было необходимо реализовать функцию, которая сможет определить секретный ключ по известным открытому и шифро текстам.

```
def key_find(open_text: str, cipher_text: str) -> list:
    #Приводим открытый и шифро тексты к бинарному виду
    cipher_text_bin = ''.join(format(ord(x), '08b') for x in cipher_text)
    open_text_bin = ''.join(format(ord(x), '08b') for x in open_text)
    #Подбираем секретный ключ применяя XOR операцию к последовательностям битов открыто
    key = []
    for idx, open_bit in enumerate(open_text_bin):
        key.append(int(cipher_text_bin[idx]) ^ int(open_bit))

    res_cipher_text, key = encrypt(open_text, key)
    assert res_cipher_text == cipher_text
    #Возвращаем полученный текст
```

Функция key_find

То, как она работает:

```
    #Examples
    key = key_find(open_text, cipher_text)
    print('Полученный серкетный ключ: {}'.format(''.join([str(x) for x in key])))
    print(f'Открытый текст: {open_text}')
    print(f'Шифротекст: {cipher_text}')
    #Применяем полученный ключ к открытому тексту, должны получить такой же шифротекст
    cipher_text, key = encrypt(open_text, key)
    print(f'Полученный шифротекст: {cipher_text}')
```

✓ 0.0s

```
Полученный серкетный ключ: 10110011111001001001010011111101100111010110110101000110010001101
Открытый текст: Happy New Year, my friends!
Шифротекст: û▯ä▯ä#ÙÀ@aå▯▯á▯oáLù▯L"Ü~]
Полученный шифротекст: û▯ä▯ä#ÙÀ@aå▯▯á▯oáLù▯L"Ü~]
```

Figure 4: Функция key_find()

Выводы

Выводы

Я узнал о схеме однократного гаммирования и научился ее применять на практике