Лабораторная работа №7

Основы информационной безопасности

Волгин Иван Алексеевич

Содержание

1 Цель работы

Освоить на практике применение режима однократного гаммирования.

2 Задание

Нужно подобрать ключ, чтобы получить сообщение «С Новым Годом, друзья!». Требуется разработать приложение, позволяющее шифровать и дешифровать данные в режиме однократного гаммирования. Приложение должно: 1. Определить вид шифротекста при известном ключе и известном открытом тексте. 2. Определить ключ, с помощью которого шифротекст может быть преобразован в некоторый фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста.

3 Теоретическое введение

Предложенная Г. С. Вернамом так называемая «схема однократного использования (гаммирования)» является простой, но надёжной схемой шифрования данных. Гаммирование представляет собой наложение (снятие) на открытые (зашифрованные) данные последовательности элементов других данных, полученной с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных. Иными словами, наложение гаммы — это сложение её элементов с элементами открытого (закрытого) текста по некоторому фиксированному модулю, значение которого представляет собой известную часть алгоритма шифрования. В соответствии с теорией криптоанализа, 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. Наложение гаммы по сути представляет собой выполнение операции сложения по модулю 2 (XOR) (обозначаемая знаком ⊠) между элементами гаммы и элементами подлежащего сокрытию текста. Напомним, как работает операция XOR над битами: 0 ⊠ 0 = 0, 0 ⊠ 1 = 1, 1 ⊠ 0 = 1, 1 ⊠ 1 = 0. 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. Если известны ключ и открытый текст, то задача нахождения шифротекста заключается в применении к каждому символу открытого текста следующего правила: Ci = Pi ⊠ Ki , (7.1) где Ci — i-й символ получившего-

ся зашифрованного послания, Pi — i-й символ открытого текста, Ki — i-й символ ключа, i = 1, m. Размерности открытого текста и ключа должны совпадать, и полученный шифротекст будет такой же длины. Если известны шифротекст и открытый текст, то задача нахождения ключа решается также в соответствии с (7.1), а именно, обе части равенства необходимо сложить по модулю 2 с Pi : Ci ⊠ Pi = Pi ⊠ Ki ⊠ Pi = Ki , Ki = Ci ⊠ Pi . Открытый текст имеет символьный вид, а ключ — шестнадцатеричное представление. Ключ также можно представить в символьном виде, воспользовавшись таблицей ASCII-кодов. К. Шеннон доказал абсолютную стойкость шифра в случае, когда однократно используемый ключ, длиной, равной длине исходного сообщения, является фрагментом истинно случайной двоичной последовательности с равномерным законом распределения. 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. Необходимые и достаточные условия абсолютной стойкости шифра: – полная случайность ключа; – равенство длин ключа и открытого текста; – однократное использование ключа. Рассмотрим пример. Ключ Центра: 05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54 Сообщение Центра: Штирлиц – Вы Герой!! D8 F2 E8 F0 EB E8 F6 20 2D 20 C2 FB 20 C3 E5 F0 EE E9 21 21 Зашифрованный текст, находящийся у Мюллера: DD FE FF 8F E5 A6 C1 F2 B9 30 CB D5 02 94 1A 38 E5 5B 51 75 Дешифровальщики попробовали ключ: 05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 55 F4 D3 07 BB BC 54 и получили текст: D8 F2 E8 F0 EB E8 F6 20 2D 20 C2 FB 20 C1 EE EB E2 E0 ED 21 Штирлиц - Вы Болван! Другие ключи дадут лишь новые фразы, пословицы, стихотворные строфы, словом, всевозможные тексты заданной длины.

4 Выполнение лабораторной работы

1. Для начала создаю функцию, которая будет генерировать случайный ключ (рис. 4.1).

```
def hex_key_gen(text):
    key = ''
    for i in range(len(text)):
        key += random.choice(string.ascii_letters + string.digits)
    return key
```

Рис. 4.1: генерация случайного ключа

2. Затем пишу функцию для шифрования и дешифрования текста (рис. 4.2).

```
def crypt(text,key):
    new_text = ''
    for i in range(len(text)):
        new_text += chr(ord(text[i]) ^ ord(key[i % len(key)]))
    return new_text
```

Рис. 4.2: шифрование текста

3. После пишу функцию, которая будет находить различные возможные ключи для определенного фрагмента, с помощью которых шифротекст может быть преобразован в фрагмент (рис. 4.3).

```
def find_key(text, fragment):
    possible_keys = []
    for i in range(len(text) - len(fragment) + 1):
        possible_key = ""
        for j in range(len(fragment)):
            possible_key += chr(ord(text[i+j]) ^ ord(fragment[j]))
        possible_keys.append(possible_key)
    return possible_keys
```

Рис. 4.3: поиск ключей

4. После этого проверяю работу всех функций, все работает корректно (рис. 4.4).

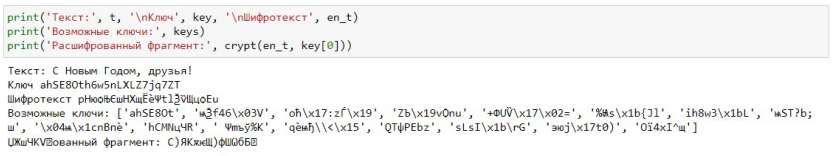

Рис. 4.4: проверка работы кода

Листинг программы

```
import random
import string

def hex_key_gen(text):
    key = ''
    for i in range(len(text)):
        key += random.choice(string.ascii_letters + string.digits)
    return key

def crypt(text,key):
    new_text = ''
    for i in range(len(text)):
        new_text += chr(ord(text[i]) ^ ord(key[i % len(key)]))
    return new_text

def find_key(text, fragment):
    possible_keys = []
    for i in range(len(text) - len(fragment) + 1):
        possible_key = ""
        for j in range(len(fragment)):
```

```
            possible_key += chr(ord(text[i+j]) ^ ord(fragment[j]))
        possible_keys.append(possible_key)
    return possible_keys

t = 'С Новым Годом, друзья!'
key = hex_key_gen(t)
en_t = crypt(t,key)
de_t = crypt(en_t,key)
keys = find_key(en_t, 'С Новым')
fragment = "С Новым"

print('Текст:', t, '\nКлюч', key, '\nШифротекст', en_t)
print('Возможные ключи:', keys)
print('Расшифрованный фрагмент:', crypt(en_t, key[0]))
```

5 Выводы

Я освоила на практике применение режима однократного гаммирования.