

Лабораторная работа №7

Дисциплина: Основы информационной безопасности

Феоктистов Владислав Сергеевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Однакратное гаммирование	7
3.2	Таблицы	9
4	Выполнение лабораторной работы	11
5	Выводы	17
6	Контрольные вопросы	18
	Список литературы	21

Список иллюстраций

3.1	Схема однократного использования Вернама	7
4.1	Запуск текстового редактора	11
4.2	Функция однократного гаммирования	11
4.3	Функция представления строки в шестнадцатичном виде . . .	12
4.4	Код проверки функций гаммирования и преобразования строки .	12
4.5	Код программы. Первая версия	12
4.6	Запуск программы. Первая версия	13
4.7	Функция генерации желаемого ключа	14
4.8	Импорт функции randint	14
4.9	Функция генерации случайной строки	14
4.10	Код проверки работы всех функций	15
4.11	Код программы. Вторая версия	15
4.12	Код программы. Вторая версия	16
4.13	Запуск программы. Вторая версия	16

Список таблиц

3.1	Описание некоторых каталогов файловой системы GNU Linux . .	9
3.2	Описание некоторых используемых в работе команд	10

1 Цель работы

Целью данной работы является освоение 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

3.1 Однократное гаммирование

Предложенная Г. С. Вернамом так называемая «схема однократного использования (гаммирования)» (рис. 3.1) является простой, но надёжной схемой шифрования данных.



Рис. 3.1: Схема однократного использования Вернама

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 \oplus) между элементами гаммы и элементами подлежащего сокрытию текста. Напомним, как работает операция XOR над битами: $0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0$.

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:

$$C_i = P_i \oplus K_i,$$

где C_i — i -й символ получившегося зашифрованного послания, P_i — i -й символ открытого текста, K_i — i -й символ ключа, $i = \overline{1, m}$. 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 $C_i = P_i \oplus K_i$, а именно, обе части равенства необходимо сложить по модулю 2 с P_i :

$$C_i \oplus P_i = P_i \oplus K_i \oplus P_i = K_i$$

$$K_i = C_i \oplus P_i$$

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 однократ-

но 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.

3.2 Таблицы

Таблица 3.1: Описание некоторых каталогов файловой системы GNU Linux

Имя каталога	Описание каталога
/	Корневая директория, содержащая всю файловую
/bin	Основные системные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям
/etc	Общесистемные конфигурационные файлы и файлы конфигурации установленных программ
/home	Содержит домашние директории пользователей, которые, в свою очередь, содержат персональные настройки и данные пользователя
/media	Точки монтирования для сменных носителей
/root	Домашняя директория пользователя root
/tmp	Временные файлы

Имя каталога	Описание каталога
/usr	Вторичная иерархия для данных пользователя

Таблица 3.2: Описание некоторых используемых в работе команд

Команда	Описание команды
touch	Создает текстовый файл по указанному пути и с указанным именем внутри пути.
nano	Запуск в терминале текстовый редактор
python3	Компилятор языка программирования python. В работе используется для запуска .py программ.

Более подробно об Unix см. в [1–6].

4 Выполнение лабораторной работы

Напишем программу, реализующую режим одноразового гаммирования, в домашнем каталоге с помощью текстового редактора *nano* [**cmd:** `nano Lab7.py`] (рис. 4.1). В случае отсутствия файла, где будет происходить написание программы, *nano* автоматически его создаст.

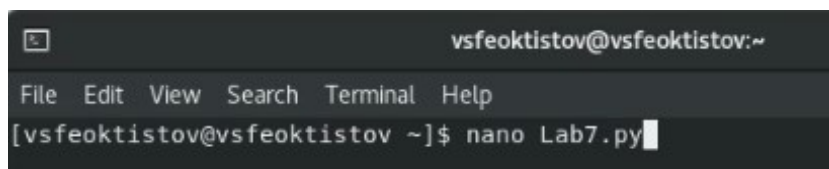


Рис. 4.1: Запуск текстового редактора

Напишем функцию, позволяющую шифровать и дешифровать текстовые сообщения в режиме одноразового гаммирования (рис. 4.2).

```
def gamming(text, key):
    """
    Функция одноразового 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 (шифрованию)
    :param key: ключ для одноразового 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))
```

Рис. 4.2: Функция одноразового гаммирования

Также напишем функцию, представляющую исходную строку в виде строки последовательности символов в шестнадцатичном виде, разделенных двоеточиями (рис. 4.3). Это необходимо, поскольку ключ имеет шестнадцатичное представление. Также это необходимо будет еще и потому, что в процессе поэле-

ментного применения XOR мы получаем новый код символа, который зачастую невозможно отобразить в нормальном виде.

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

Рис. 4.3: Функция представления строки в шестнадцатиричном виде

Проверим работу, написанных нами, функций. В качестве открытого текста для шифрования используем сообщение “Hello world!”, а в качестве ключа сообщение “Just message” (ключ может быть и случайным набором символов той же длины, что и открытый текст). Проверку будем осуществлять в блоке *if name == 'main':*, который отвечает за входную функцию при запуске программы (рис. 4.4).

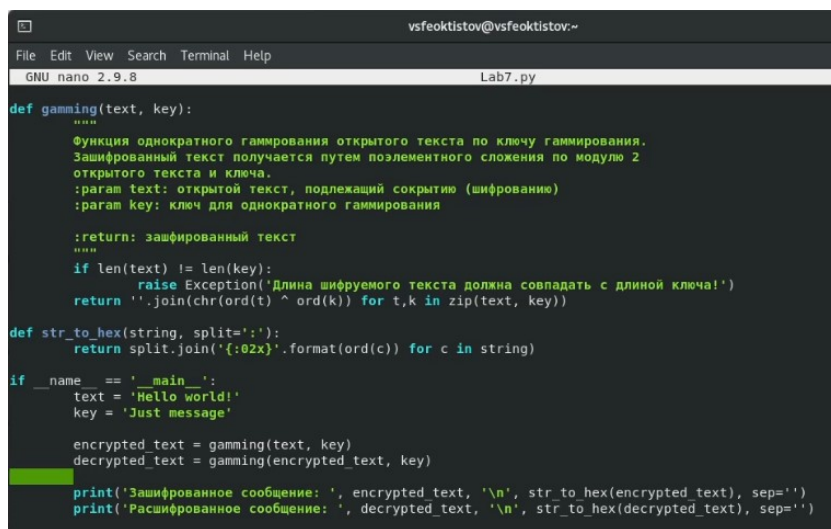
```
if __name__ == '__main__':
    text = 'Hello world!'
    key = 'Just message'

    encrypted_text = gamming(text, key)
    decrypted_text = gamming(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='')
```

Рис. 4.4: Код проверки функций гаммирования и преобразования строки

В итоге, весь код будет выглядеть следующим образом (рис. 4.5):



```
def gamming(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 (шифрованию)
    :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'

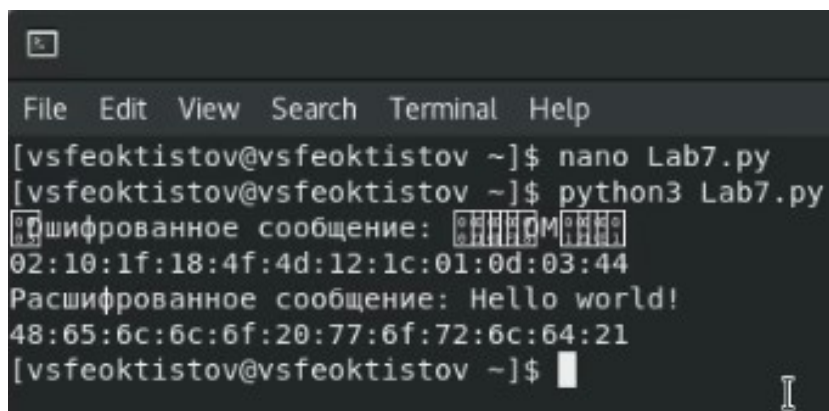
    encrypted_text = gamming(text, key)
    decrypted_text = gamming(encrypted_text, key)

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

Рис. 4.5: Код программы. Первая версия

Запустим программу, для этого сохраним код и выйдем из редактора *nano*

(сочетание клавиш *Ctrl* + *X* + ввод символа 'у' для подтверждения изменений + нажатие клавиши 'Enter'), после чего с помощью команды *python3 Lab7.py* запустим её (рис. 4.6).



```
File Edit View Search Terminal Help
[vsfeoktistov@vsfeoktistov ~]$ nano Lab7.py
[vsfeoktistov@vsfeoktistov ~]$ python3 Lab7.py
Зашифрованное сообщение: 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 ~]$
```

Рис. 4.6: Запуск программы. Первая версия

Как можно заметить, однократное гаммирование открытого текста приводит к его шифрованию, причем получается набор символов, которые не получается отобразить в нормальном виде в консоли (т.к. кодировка в консоли не имеет конкретное символьное представление для них), однако можно увидеть шестнадцатичное представление зашифрованного текста; однократное гаммирование зашифрованного текста тем же ключом приводит к его расшифровке. Важно отметить, что на самом деле порядок задания параметров в функции *gamming(text, key)* не имеет значение, т.е. вместо *key* можно подставить *text*, а вместо *text* - *key*. Это возможно, поскольку операция сложения по модулю 2 - коммутативна.

Далее расширим функционал программы: напишем функцию генерации необходимого ключа и функцию генерации случайно строки заданной длины.

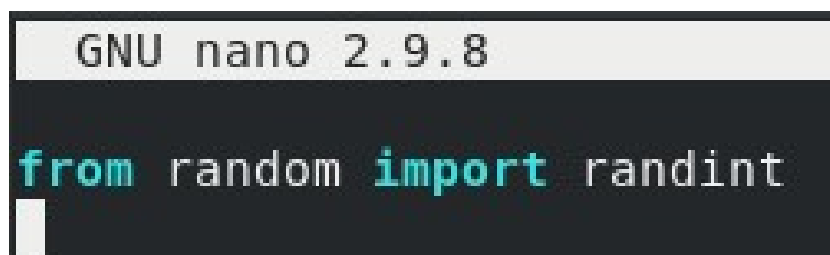
Для начала напишем функцию генерации ключа для указанного текста, которое будет после однократного гаммирование этого текста этим ключом выдавать необходимый текст (рис. 4.7).

```
def get_key(text, required_text):
    """
    Функция нахождения ключа, дающего желаемый текст, после
    однократного гаммирования открытого сообщения.
    :param text: открытый текст, который в дальнейшем можно
    будет зашифровать однократным гаммированием найденным ключом
    :param required_text: желаемый текст, который нужно получить
    после однократного гаммирования указанного сообщения найденным ключом
    :return: желаемый ключ гаммирования
    """
    if len(text) != len(required_text):
        raise Exception('Длина шифруемого текста должна совпадать с длиной желаемого текста после однократного гаммирования!')
    return gamming(text, required_text)
```

Рис. 4.7: Функция генерации желаемого ключа

Как можно заметить, по факту эта функция является оберткой функции *gamming*. Однако, эта функция удобнее в плане говорящего имени и параметров, а также в связи с друим выводом ошибки в случае, если длины строк, задаваемых в параметрах, - разные.

Для удобства дальнейшей работы, напомним функцию генерации случайной строки заданной длины *length* с использованием заданных символов *letters*, причем по умолчанию - это латинские символы верхнего и нижнего регистров (рис. 4.9). Но перед этим нужно импортировать необходимую функцию из библиотеки *random* (рис. 4.8).



```
GNU nano 2.9.8
from random import randint
```

Рис. 4.8: Импорт функции randint

```
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 использовать 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))
```

Рис. 4.9: Функция генерации случайной строки

Теперь же можно переписать содержимое точки вхождения программы (усло-

вие *if name == 'main':*). Открытый текст теперь будет содержать сообщение “С Днем народного единства!”, а ключ гаммирования мы сгенерируем случайным образом, с помощью функции генерации случайно строки. В результате получим зашифрованный текст после первого гаммирования и расшифрованный - после второго гаммирования. Также добавим переменную с желаемым текстом, получаемого после однократного гаммирования указанного сообщения, - “С Новым годом, друзья!”. Пробелы в конце были добавлены, чтобы размеры сообщений и ключей совпадали. В процессе работы программы, все сообщения и ключи будут выведены в символьном и шестнадцатиричном видах.

```
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 = gamming(text, key)
    decrypted_text = gamming(encrypted_text, key)

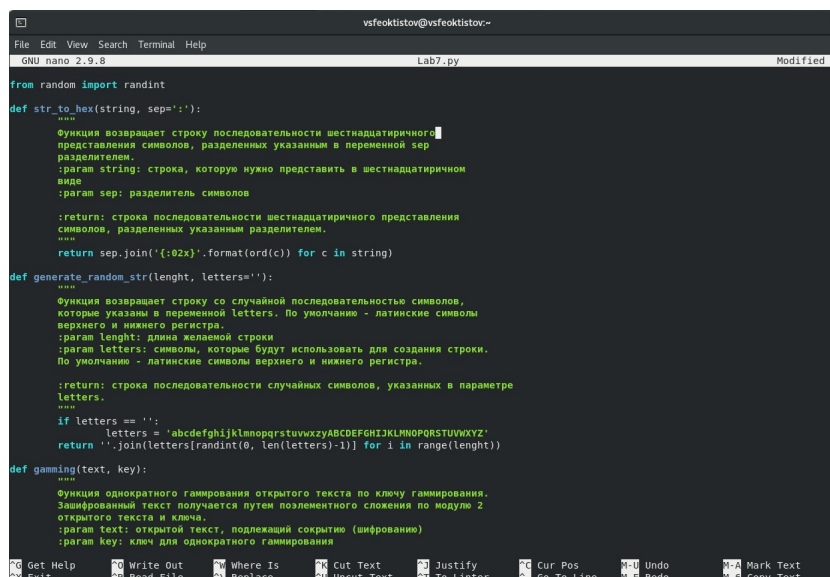
    print('Зашифрованное сообщение: ', encrypted_text, '\n', str_to_hex(encrypted_text), sep='')
    print('Расшифрованное сообщение: ', decrypted_text, '\n', str_to_hex(decrypted_text), sep='')

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

Рис. 4.10: Код проверки работы всех функций

В итоге, весь код будет выглядеть следующим образом (рис. 4.11 и 4.12):



```
from random import randint

def str_to_hex(string, sep=':'):
    """
    Функция возвращает строку последовательности шестнадцатиричного
    представления символов, разделенных указанным в переменной sep
    разделителем.
    :param string: строка, которую нужно представить в шестнадцатиричном
    виде
    :param sep: разделитель символов
    :return: строка последовательности шестнадцатиричного представления
    символов, разделенных указанным разделителем.
    """
    return sep.join('{:02x}'.format(ord(c)) for c in string)

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 = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
    return ''.join(letters[randint(0, len(letters)-1)] for i in range(length))

def gamming(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 (шифрованию)
    :param key: ключ для однократного гаммирования
    """
```

Рис. 4.11: Код программы. Вторая версия

```

vsfeoktistov@vsfeoktistov-
File Edit View Search Terminal Help
GNU nano 2.9.8 Lab7.py Modified

: 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 get_key(text, required_text):
    """
    Функция нахождения ключа, дающего желаемый текст, после
    однократного гаммирования открытого сообщения.
    :param text: открытый текст, который в дальнейшем можно
    будет зашифровать однократным гаммированием найденным ключом
    :param required_text: желаемый текст, который нужно получить
    после однократного гаммирования указанного сообщения найденным ключом

    : return: желаемый ключ гаммирования
    """
    if len(text) != len(required_text):
        raise Exception('Длина шифруемого текста должна совпадать с длиной желаемого текста после однократного гаммирования!')
    return gamming(text, required_text)

if __name__ == '__main__':
    text = 'С Днем народного единства!' # len = 20
    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 = gamming(text, key)
    decrypted_text = gamming(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 = gamming(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))

Get Help Write Out Where Is Cut Text Justify Cur Pos Undo Mark Text
Exit Read File Replace Uncut Text To Linter Go To Line Redo Copy Text

```

Рис. 4.12: Код программы. Вторая версия

Запустим программу и проверим ее работу [cmd: python3 Lab7.py] (рис. 4.13).

```

vsfeoktistov@vsfeoktistov-
File Edit View Search Terminal Help
[vsfeoktistov@vsfeoktistov ~]$ nano Lab7.py
[vsfeoktistov@vsfeoktistov ~]$ python3 Lab7.py
Исходный открытый текст: С Днем народного единства!
421:20:414:43d:435:43c:20:43d:430:440:43e:434:43d:43e:433:43e:20:435:434:438:43d:441:442:432:430:21
Ключ гаммирования: GrvGBJHmzYULTbMcOhJtSkLIYo
47:72:76:47:42:4a:48:6d:7a:59:5b:6c:54:62:4d:63:4f:68:4a:74:53:6b:6c:49:59:6f
Зашифрованное сообщение: aRBoYrHnZjKjKzKjOjZj30==N
466:52:462:47a:477:476:68:450:44a:419:46b:458:469:45c:47e:45d:6f:45d:47e:44c:46e:42a:42e:47b:469:4e
Расшифрованное сообщение: С Днем народного единства!
421:20:414:43d:435:43c:20:43d:430:440:43e:434:43d:43e:433:43e:20:435:434:438:43d:441:442:432:430:21
Желаемый ключ: GrDE=evY' fUvYjRj3fYUшп
47:72:7f:44:45:3d:454:470:59:27:5f:66:55:470:45e:69:42f:1e:49:00:21:40b:40e:45b:449:6e
Расшифровка сообщения ключом, которое дает желаемое сообщение для известного шифротекста: С Новым Годом, друзья!
421:20:41d:43e:432:44b:43c:20:413:43e:434:43e:43c:2c:20:434:440:443:437:44c:44f:21:20:20:20:20:2
[vsfeoktistov@vsfeoktistov ~]$

```

Рис. 4.13: Запуск программы. Вторая версия

Как видно из рисунка выше, использование повторного гаммирования тем же ключом приводит к расшифровке (расшифрованное сообщение полностью совпадает с сообщением до шифрования). Для зашифрованного сообщения можно подобрать ключ, который после применения однократного гаммирования даст желаемое сообщение, отличное от начального (сообщения до шифрования). Такой же ключ можно сгенерировать и для открытого текста, например, чтобы запутать человека, которому не было адресовано настоящее сообщение. В результате применения такого ключа для зашифрованного сообщения, мы получили сообщение “С Новым Годом, друзья!”.

5 Выводы

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

6 Контрольные вопросы

1. Поясните смысл однократного гаммирования.

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).

2. Перечислите недостатки однократного гаммирования.

Недостатки однократного гаммирования:

- ключ, используемый для шифрования/дешифрования, нельзя передать по открытой сети;
- ключ зачастую должен совпадать с длиной шифруемого/дешифруемого текста, а зацикленный ключ легко вычислить;
- поэлементное шифрование без зависимости от предыдущего символа или текста в целом не обеспечивает надлежащую безопасность.

3. Перечислите преимущества однократного гаммирования.

Преимущества однократного гаммирования:

- простой принцип работы (это как плюс, так и минус);
- работает очень быстро, так как нет сложных вычислений и зависимостей;
- подбор ключа, с помощью которого можно зашифровать текст так, чтобы он казался похожим на исходный, но с другими данными, либо не представляющий интереса текст. Тем самым можно запутать третье лицо, котором не было адресовано сообщение.

4. Почему длина открытого текста должна совпадать с длиной ключа?

Это требуется, поскольку в процессе однократного гаммирования происходит поэлементное сложение по модулю 2 открытого текста и ключа. Если длина ключа меньше длины открытого текста, то можно будет зашифровать только его часть, иначе же, нужно будет обрезать часть ключа до нужного размера.

5. Какая операция используется в режиме однократного гаммирования, назовите её особенности?

В режиме однократного гаммирования происходит поэлементное сложение по модулю 2 (XOR) (обозначаемая знаком \oplus).

Особенность XOR заключается в том, что одной и той же функцией можно как зашифровать, так и расшифровать их (двойное прибавление одной и той же величины по модулю 2 восстанавливает исходное).

6. Как по открытому тексту и ключу получить шифротекст?

Если известны ключ и открытый текст, то задача нахождения шифротекста заключается в применении к каждому символу открытого текста следующего правила:

$$C_i = P_i \oplus K_i,$$

где C_i — i -й символ получившегося зашифрованного послания, P_i — i -й символ открытого текста, K_i — i -й символ ключа, $i = \overline{1, m}$. Размерности открытого текста и ключа должны совпадать, и полученный шифротекст будет такой же длины.

7. Как по открытому тексту и шифротексту получить ключ?

Если известны шифротекст и открытый текст, то задача нахождения ключа решается также в соответствии с $C_i = P_i \oplus K_i$, а именно, обе части равенства необходимо сложить по модулю 2 с P_i :

$$C_i \oplus P_i = P_i \oplus K_i \oplus P_i = K_i$$

$$K_i = C_i \oplus P_i$$

где C_i — i -й символ получившегося зашифрованного послания, P_i — i -й символ открытого текста, K_i — i -й символ ключа, $i = \overline{1, m}$. Размерности открытого текста и ключа должны совпадать, и полученный шифротекст будет такой же длины.

8. В чем заключаются необходимые и достаточные условия абсолютной стойкости шифра?

Необходимые и достаточные условия абсолютной стойкости шифра:

- полная случайность ключа (ключ не должен состоять из одного и того же символа или зацикливаться, иначе его будет легко расшифровать);
- равенство длин ключа и открытого текста (поэлементные операции требуют равенства длин);
- однократное использование ключа (повторное использование приводит к возвращению к исходному тексту/последовательности битов).

Список литературы

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016.
URL: <https://www.gnu.org/software/bash/manual/>.
2. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.