

Лабораторная работа №8

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
4.1	Написание вспомогательных функций	11
4.2	Эмуляция сценария расшифровки злоумышленником двух сообщений	13
5	Выводы	17
6	Контрольные вопросы	18
	Список литературы	21

Список иллюстраций

3.1	Схема однократного использования Вернама	7
4.1	Импорт функции randint	11
4.2	Функция добавления пробелов	11
4.3	Функция представления строки в шестнадцатиричном виде . . .	12
4.4	Функция генерации случайной строки	12
4.5	Функция однократного гаммирования	13
4.6	Код эмуляции ситуации взлома	14
4.7	Работа программы расшифровки	15
4.8	Работа программы расшифровки	15
6.1	Общая схема шифрования двух различных текстов одним ключом	19

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 на примере кодирования различных исходных текстов одним ключом.

2 Задание

Два текста кодируются одним ключом (однократное гаммирование). Требуется не зная ключа и не стремясь его определить, прочесть оба текста. Необходимо разработать приложение, позволяющее шифровать и дешифровать тексты P_1 и P_2 в режиме однократного гаммирования. Приложение должно определить вид шифротекстов C_1 и C_2 обоих текстов P_1 и P_2 при известном ключе ; Необходимо определить и выразить аналитически способ, при котором злоумышленник может прочесть оба текста, не зная ключа и не стремясь его определить.

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.

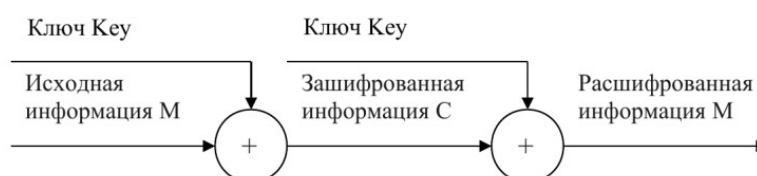


Рис. 3.1: Схема однократного использования Вернама

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

4.1 Написание вспомогательных функций

Для начала импортируем необходимые для работы библиотеки и функции (рис. 4.1):

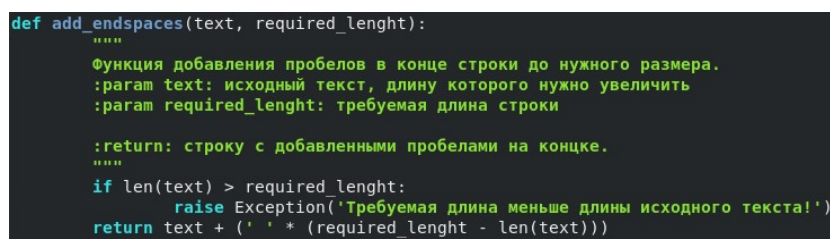


```
GNU nano 2.9.8

from random import randint
```

Рис. 4.1: Импорт функции randint

Напишем функцию добавления пробелов в конце строки до нужного размера строки (рис. 4.3). Это необходимо, поскольку длины открытых сообщений должны совпадать, иначе их нельзя будет зашифровать одним и тем же ключом гаммирования.



```
def add_endspaces(text, required_lenght):
    """
    Функция добавления пробелов в конце строки до нужного размера.
    :param text: исходный текст, длину которого нужно увеличить
    :param required_lenght: требуемая длина строки

    :return: строку с добавленными пробелами на конце.
    """
    if len(text) > required_lenght:
        raise Exception('Требуемая длина меньше длины исходного текста!')
    return text + (' ' * (required_lenght - len(text)))
```

Рис. 4.2: Функция добавления пробелов

Также напишем функцию, представляющую исходную строку в виде строки последовательности символов в шестнадцатичном виде, разделенных двоеточием.

точиями (рис. 4.2). Это необходимо, поскольку ключ имеет шестнадцатиричное представление. Также это необходимо будет еще и потому, что в процессе поэлементного применения XOR мы получаем новый код символа, который зачастую невозможно отобразить в нормальном виде.

```
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)
```

Рис. 4.3: Функция представления строки в шестнадцатиричном виде

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. 4.4).

```
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))
```

Рис. 4.4: Функция генерации случайной строки

Напишем функцию, позволяющую шифровать и дешифровать текстовые сообщения в режиме однократного гаммирования (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))
```

Рис. 4.5: Функция однократного гаммирования

4.2 Эмуляция сценария расшифровки злоумышленником двух сообщений

Пусть у центра есть две телеграммы P_1 и P_2 , а также ключ K . Тогда с помощью этого ключа можно зашифровать эти две телеграммы методом однократного гаммирования:

$$C_1 = P_1 \oplus K$$

$$C_2 = P_2 \oplus K$$

Открытый текст можно найти в соответствии с формулами выше, зная шифротекст двух телеграмм, зашифрованных одним ключом. Для это оба равенства складываются по модулю 2. Тогда с учётом свойства операции XOR

$$1 \oplus 1 = 0, 1 \oplus 0 = 1$$

получаем:

$$C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$$

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 $C_1 \oplus C_2$ (известен вид обеих шифровок). Тогда зная P_1 и учитывая $C_1 \oplus C_2 = P_1 \oplus P_2$, имеем:

$$C_1 \oplus C_2 \oplus P_1 = P_1 \oplus P_2 \oplus P_1 = P_2$$

Таким образом, злоумышленник получает возможность определить те символы сообщения P_2 , которые находятся на позициях известного шаблона сообщения P_1 . В соответствии с логикой сообщения P_2 , злоумышленник имеет реальный шанс узнать ещё некоторое количество символов сообщения P_2 . Затем вновь используется та же формула с подстановкой вместо P_1 полученных на предыдущем шаге новых символов сообщения P_2 . И так далее. Действуя подобным образом, злоумышленник даже если не прочтает оба сообщения, то значительно уменьшит пространство их поиска.

Сымитируем похожую ситуацию, используя реализованные выше функции (рис. 4.6).

```
if __name__ == '__main__':
    P1 = 'Новости: В московском метро появился новый тематический поезд "Сибирь здесь"'
    P2 = 'Южный речной вокзал в Москве планирует открыть к лету 2023 года'

    P1_hacked = 'Новости: '
    P2_hacked = ''
    k = 0
    prev_k = k

    lenght = max(len(P1), len(P2))
    P1 = add_endspaces(P1, lenght)
    P2 = add_endspaces(P2, lenght)
    K = generate_random_str(lenght)

    C1 = gamming(P1, K)
    C2 = gamming(P2, K)
    C = gamming(C1, C2)

    while len(P1_hacked) != lenght and len(P2_hacked) != lenght:
        if k % 2 == 0:
            P2_hacked_tmp = gamming(C[:len(P1_hacked)], P1_hacked)
            print('\nУдалось расшифровать часть второго сообщения:', P2_hacked_tmp)
            if k != prev_k:
                if input('Back? [y/n]: ') == 'y':
                    k -= 1
                    P1_hacked = P1_hacked_prev
                    continue
            P2_hacked = P2_hacked_tmp
            P2_hacked_prev = P2_hacked_tmp
            P2_hacked += input('Введите предполагаемое продолжение для второго сообщения: ')
            print('Теперь второе сообщение имеет вид: ', P2_hacked)
        else:
            P1_hacked_tmp = gamming(C[:len(P2_hacked)], P2_hacked)
            print('\nУдалось расшифровать часть первого сообщения:', P1_hacked_tmp)
            if k != prev_k:
                if input('Back? [y/n]: ') == 'y':
                    k -= 1
                    P2_hacked = P2_hacked_prev
                    continue
            P1_hacked = P1_hacked_tmp
            P1_hacked_prev = P1_hacked_tmp
            P1_hacked += input('Введите предполагаемое продолжение для первого сообщения: ')
            print('Теперь первое сообщение имеет вид: ', P1_hacked)
        if input('Continue? [y/n]: ') == 'n':
            break
        prev_k = k
        k += 1

    if len(P1_hacked) == lenght and len(P2_hacked) == lenght:
        print('Для сообщения были полностью расшифрованы!')
```

Рис. 4.6: Код эмуляции ситуации взлома

Теперь можно посмотреть, как работает программа [cmd: python3 Lab8.py] (рис.

4.7 и 4.8):

```
vsfeoktistov@vsfeoktistov:~$ nano Lab8.py
vsfeoktistov@vsfeoktistov:~$ python3 Lab8.py

Удалось расшифровать часть второго сообщения: Южный реч
Введите предполагаемое продолжение для второго сообщения: ной вокзал
Теперь второе сообщение имеет вид: Южный речной вокзал
Continue? [y/n]: y

Удалось расшифровать часть первого сообщения: Новости: В московск
Back? [y/n]: n
Введите предполагаемое продолжение для первого сообщения: ом зоопарке
Теперь первое сообщение имеет вид: Новости: В московском зоопарке
Continue? [y/n]: y

Удалось расшифровать часть второго сообщения: Южный речной вокзал в ЗенхмУдТ
Back? [y/n]: y

Удалось расшифровать часть первого сообщения: Новости: В московск
Введите предполагаемое продолжение для первого сообщения: ом метро
Теперь первое сообщение имеет вид: Новости: В московском метро
Continue? [y/n]: y

Удалось расшифровать часть второго сообщения: Южный речной вокзал в Москве
Back? [y/n]: n
Введите предполагаемое продолжение для второго сообщения: планируют
Теперь второе сообщение имеет вид: Южный речной вокзал в Москве планируют
Continue? [y/n]: y

Удалось расшифровать часть первого сообщения: Новости: В московском метро появился н
Back? [y/n]: n
Введите предполагаемое продолжение для первого сообщения: овый
Теперь первое сообщение имеет вид: Новости: В московском метро появился новый
Continue? [y/n]: y

Удалось расшифровать часть второго сообщения: Южный речной вокзал в Москве планируют откр
Back? [y/n]: n
Введите предполагаемое продолжение для второго сообщения: ыть
Теперь второе сообщение имеет вид: Южный речной вокзал в Москве планируют открыть
Continue? [y/n]: y

Удалось расшифровать часть первого сообщения: Новости: В московском метро появился новый тема
Back? [y/n]: n
Введите предполагаемое продолжение для первого сообщения: тический
Теперь первое сообщение имеет вид: Новости: В московском метро появился новый тематический
Continue? [y/n]: y
```

Рис. 4.7: Работа программы расшифровки

```
vsfeoktistov@vsfeoktistov:~$ python3 Lab8.py

Удалось расшифровать часть второго сообщения: Южный речной вокзал в Москве планируют откр
Back? [y/n]: n
Введите предполагаемое продолжение для второго сообщения: ыть
Теперь второе сообщение имеет вид: Южный речной вокзал в Москве планируют открыть
Continue? [y/n]: y

Удалось расшифровать часть первого сообщения: Новости: В московском метро появился новый тема
Back? [y/n]: n
Введите предполагаемое продолжение для первого сообщения: тический
Теперь первое сообщение имеет вид: Новости: В московском метро появился новый тематический
Continue? [y/n]: y

Удалось расшифровать часть второго сообщения: Южный речной вокзал в Москве планируют открыть к лету 20
Back? [y/n]: n
Введите предполагаемое продолжение для второго сообщения: 23 года
Теперь второе сообщение имеет вид: Южный речной вокзал в Москве планируют открыть к лету 2023 года
Continue? [y/n]: y

Удалось расшифровать часть первого сообщения: Новости: В московском метро появился новый тематический поезд "
Back? [y/n]: n
Введите предполагаемое продолжение для первого сообщения: "
Теперь первое сообщение имеет вид: Новости: В московском метро появился новый тематический поезд ""
Continue? [y/n]: y

Удалось расшифровать часть второго сообщения: Южный речной вокзал в Москве планируют открыть к лету 2023 годаУ
Back? [y/n]: y

Удалось расшифровать часть первого сообщения: Новости: В московском метро появился новый тематический поезд "
Введите предполагаемое продолжение для первого сообщения:
Теперь первое сообщение имеет вид: Новости: В московском метро появился новый тематический поезд "
Continue? [y/n]: y

Удалось расшифровать часть второго сообщения: Южный речной вокзал в Москве планируют открыть к лету 2023 года
Back? [y/n]: n
Введите предполагаемое продолжение для второго сообщения:
Теперь второе сообщение имеет вид: Южный речной вокзал в Москве планируют открыть к лету 2023 года
Continue? [y/n]: y

Удалось расшифровать часть первого сообщения: Новости: В московском метро появился новый тематический поезд "Сибирь з
Back? [y/n]: n
Введите предполагаемое продолжение для первого сообщения: здесь"
Теперь первое сообщение имеет вид: Новости: В московском метро появился новый тематический поезд "Сибирь здесь"
Continue? [y/n]: y
vsfeoktistov@vsfeoktistov:~$
```

Рис. 4.8: Работа программы расшифровки

Как можно видеть по рисункам выше, зная лишь только часть одного из сообщений, можно постепенно расшифровать оба сообщения. Таким образом, приходим

к выводу, что использование одного и того же ключа для шифрования нескольких сообщений методов одноразового гаммирования - это плохая идея.

5 Выводы

В процессе выполнения лабораторной работы освоил на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом; написал программу, позволяющую расшифровать два сообщения, зная только один из них.

6 Контрольные вопросы

1. Как, зная один из текстов (P_1 или P_2), определить другой, не зная при этом ключа?

Допустим, что злоумышленнику известен полностью текст P_1 . Тогда учитывая $C_1 \oplus C_2 = P_1 \oplus P_2$, имеем:

$$C_1 \oplus C_2 \oplus P_1 = P_1 \oplus P_2 \oplus P_1 = P_2$$

Таким образом, злоумышленник может полностью получить второй текст (P_2). При этом для этого даже не нужно находить ключ гаммирования. Таким же способом можно найти весь текст P_1 , если известен полностью текст P_2 . Если же известна только часть только одного текста, то можно угадывать символы постепенно, расшифровывая один за счет другого и добавляя вероятную последовательность символов в соответствии со спецификой информации и языка.

2. Что будет при повторном использовании ключа при шифровании текста?

При повторном использовании ключа при шифровании текста, в соответствии с формулами:

$$C_i = P_i \oplus K_i$$

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

получаем исходный текст (текст до шифровки).

3. Как реализуется режим шифрования однократного гаммирования одним ключом двух открытых текстов?

Режим шифрования однократного гаммирования одним ключом двух видов открытого текста реализуется в соответствии с формулами и со схемой ниже:

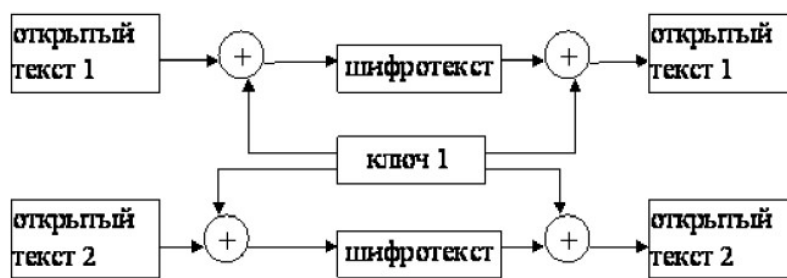


Рис. 6.1: Общая схема шифрования двух различных текстов одним ключом

Пусть у центра есть две телеграммы P_1 и P_2 , а также ключ K . Тогда с помощью этого ключа можно зашифровать эти две телеграммы методом однократного гаммирования:

$$C_1 = P_1 \oplus K$$

$$C_2 = P_2 \oplus K$$

4. Перечислите недостатки шифрования одним ключом двух открытых текстов.

Недостатки шифрования одним ключом двух открытых текстов:

- зная полностью одно сообщение, можно получить и второе;

- зная лишь часть одного сообщения, возможно постепенно расшифровать оба сообщения;
- необходимо, чтобы оба сообщения были одинаковых размеров (либо создать один ключ длиной равной максимум длин сообщений и для коротких обрезать его, тем самым можно немного увеличить безопасность).

5. Перечислите преимущества шифрования одним ключом двух открытых текстов.

Преимущества шифрования одним ключом двух открытых текстов:

- из преимуществ только немного сэкономленная память и сокращенное число итераций. Однако это полностью нивелируется отсутствием достаточной безопасности.

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.