

# Лабораторная работа №7

Элементы криптографии. Однократное гаммирование

Калинина Кристина Сергеевна

# Содержание

Цель работы	5
Теоретические сведения	6
Выполнение лабораторной работы	8
Выводы	10
Контрольные вопросы	11
Список литературы	13

# List of Figures

0.1	Блок программы с библиотеками и функциями . . . . .	8
0.2	Блок программы с выполнением первого пункта . . . . .	9
0.3	Блок программы с выполнением первого пункта . . . . .	9

# List of Tables

## Цель работы

Освоить на практике применение режима однократного гаммирования.

# Теоретические сведения

С точки зрения теории криптоанализа метод шифрования однократной случайной равновероятной гаммой той же длины, что и открытый текст, является невскрываемым (далее для краткости авторы будут употреблять термин “однократное гаммирование”, держа в уме все вышесказанное). Обоснование, которое привел Шеннон, основываясь на введенном им же понятии информации, не дает возможности усомниться в этом - из-за равных априорных вероятностей криптоаналитик не может сказать о дешифровке, верна она или нет. Кроме того, даже раскрыв часть сообщения, дешифровщик не сможет хоть сколько нибудь поправить положение - информация о вскрытом участке гаммы не дает информации об остальных ее частях. [1]

Логично было бы предположить, что для организации канала конфиденциальной связи в открытых сетях следовало бы воспользоваться именно схемой шифрования однократного гаммирования. Ее преимущества вроде бы очевидны. Есть, правда, один весомый недостаток, который сразу бросается в глаза, - это необходимость иметь огромные объемы данных, которые можно было бы использовать в качестве гаммы. Для этих целей обычно пользуются датчиками настоящих случайных чисел (в западной литературе аналогичный термин носит название True Random Number Generator или TRNG). Это уже аппаратные устройства, которые по запросу выдают набор случайных чисел, генерируя их с помощью очень большого количества физических параметров окружающей среды. Статистические характеристики таких наборов весьма близки к характеристикам “белого шума”, что означает равновероятное появление каждого следующего числа в наборе. А это, в свою очередь, означает

для нас действительно равновероятную гамму. [1]

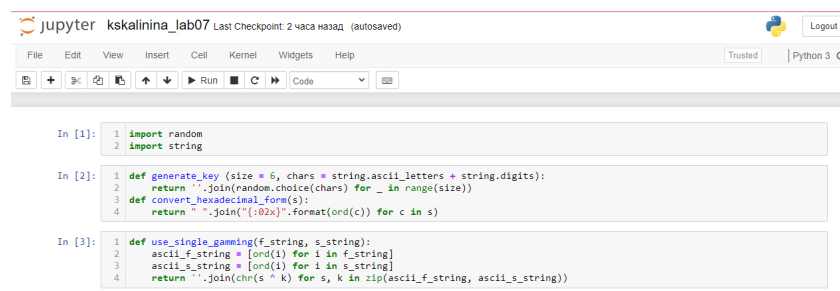
К сожалению, для того чтобы организовать конфиденциальный канал передачи данных, потребуется записать довольно большое количество этих данных и обменяться ими по секретному каналу. [1]

Уже одно это условие делает однократное гаммирование во многих случаях неприемлемым. В самом деле, зачем передавать что-то по открытому незащищенному каналу, когда есть возможность передать все это по секретному защищенному? И хотя на простой вопрос, является ли метод использования однократной случайной равновероятной гаммы стойким к взлому, существует положительный ответ, его использование может оказаться попросту невозможным. [1]

Да и к тому же метод однократного гаммирования криптостоек только в определенных, можно даже сказать, тепличных условиях. [1]

# Выполнение лабораторной работы

1. Для реализации приложения было принято решение воспользоваться jupyter notebook в котором я создала файл и на языке python написала программный код.
2. Сначала я подключила необходимые библиотеки: random для использования рандомайзера при генерации ключей и string для использования констант. Затем я написала две функции. Первая - generate\_key - генерирует ключ той же длины, что и строка, которую нужно зашифровать, принимая на вход длину этой самой строки. Функция возвращает сгенерированный ключ в виде строки. Вторая - convert\_hexadecimal\_form - выполняет перевод в шестнадцатиричную систему строку, которую принимает на вход. Функция возвращает строку в шестнадцатиричном виде (fig. 0.1).



```
In [1]: 1 import random
        2 import string

In [2]: 1 def generate_key (size = 6, chars = string.ascii_letters + string.digits):
        2     return ''.join(random.choice(chars) for _ in range(size))
        3 def convert_hexadecimal_form(s):
        4     return " ".join("{:02x}".format(ord(c)) for c in s)

In [3]: 1 def use_single_gaming(f_string, s_string):
        2     ascii_f_string = [ord(i) for i in f_string]
        3     ascii_s_string = [ord(i) for i in s_string]
        4     return ''.join(chr(s ^ k) for s, k in zip(ascii_f_string, ascii_s_string))
```

Figure 0.1: Блок программы с библиотеками и функциями

3. В первом пункте нужно было определить вид шифротекста при известном ключе и известном открытом тексте. Поэтому я ввела нужную строку. Сгенерировала для неё ключ. И вывела эту строку в зашифрованном виде. При



этом дублировала выводы в шестнадцатичном виде (fig. 0.2).

```
Задание 1
In [4]: 1 cur_string = "С Новым Годом, друзья"
        2
        3 key = generate_key(len(cur_string))
        4
        5 print("Ключ: ", key)
        6 print("Ключ(16): ", convert_hexadecimal_form(key))
        7
        8 new_string = use_single_gamming(cur_string, key)
        9
        10 print("Зашифрованная строка: ", new_string)
        11 print("Зашифрованная строка(16): ", convert_hexadecimal_form(new_string))

Ключ: yWti7j7ntZWcfRIHH88sa
Ключ(16): 79 57 74 69 37 6a 37 6e 74 5a 57 43 66 52 49 48 48 38 42 73 61
Зашифрованная строка: jwm1SCNIAkft0w~l0Jovm0
Зашифрованная строка(16): 458 77 469 457 405 421 40b 4e 467 464 463 47d 45a 7e 69 47c 408 47b 475 43f 42e
```

Figure 0.2: Блок программы с выполнением первого пункта

4. Во втором пункте нужно было определить ключ, с помощью которого шифротекст может быть преобразован в некоторый фрагмент текста, представляющий собой один из возможных вариантов прочтения открытого текста. Поэтому я сгенерировала новый ключ нужной длины и попыталась им дешифровать запись. Полученный результат вывела в исходном и шестнадцатеричном виде. Также я заново сгенерировала ключ, используя исходную и зашифрованную строки из первого пункта, и удачно расшифровала строку. (fig. 0.3).

```
Задание 2
In [7]: 1 key = generate_key(len(cur_string))
        2
        3 print("Ключ: ", key)
        4 print("Ключ(16): ", convert_hexadecimal_form(key))
        5
        6 print("Полученная строка: ", use_single_gamming(new_string, key))
        7
        8 key = use_single_gamming(cur_string, new_string)
        9 print("\nКлюч: ", key)
        10 print("Ключ(16): ", convert_hexadecimal_form(key))
        11
        12 print("Полученная строка: ", use_single_gamming(new_string, key))

Ключ: S8Pd12Xw88XAgqCJ5esJf
Ключ(16): 53 38 50 64 49 32 58 57 38 38 58 41 67 71 43 4a 53 65 73 6a 46
Полученная строка: kOWtgy77Buklmm2*ah0Is#

Ключ: yWti7j7ntZWcfRIHH88sa
Ключ(16): 79 57 74 69 37 6a 37 6e 74 5a 57 43 66 52 49 48 48 38 42 73 61
Полученная строка: С Новым Годом, друзья
```

Figure 0.3: Блок программы с выполнением первого пункта

## Выводы

Таким образом я успешно освоила на практике применение режима однократного гаммирования.

# Контрольные вопросы

1. Поясните смысл однократного гаммирования.

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.

2. Перечислите недостатки однократного гаммирования.

Нестойкость шифра при повторном использовании ключа и последовательность доступа к информации.

3. Перечислите преимущества однократного гаммирования.

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.

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

Потому что для каждого символа строки выполняется операция сложения по модулю 2 с символом ключа. Поэтому размерности открытого текста и ключа должны совпадать, и полученный шифротекст будет такой же длины.

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

Наложение гаммы по сути представляет собой выполнение операции сложения по модулю 2 (XOR) между элементами гаммы и элементами подлежащего сокрытию текста. 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.

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  $\tilde{N}_i$  —  $i$ -й символ получившегося зашифрованного послания,  $P_i$  —  $i$ -й символ открытого текста,  $K_i$  —  $i$ -й символ ключа.

7. Как по открытому тексту и шифротексту получить ключ?

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

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

- полная случайность ключа;
- равенство длин ключа и открытого текста;
- однократное использование ключа.

## Список литературы

1. Использование однократного гаммирования. // bugtraq.ru URL: <https://bugtraq.ru/library/boo> (дата обращения 11.12.2021).
2. Д. С. Кулябов, А. В. Королькова, М. Н. Геворкян. Информационная безопасность компьютерных сетей: лабораторные работы. // Факультет физико-математических и естественных наук. М.: РУДН, 2015. 64 с..