Лабораторная работа №3

Шифрование гаммированием

Сасин Ярослав Игоревич, НФИмд-01-22”

Содержание

# 1 Цель работы

Цель данной работы — изучить методы шифрования гаммированием.

# 2 Задание

Заданием является:

* Реализовать алгоритм шифрования гаммированием конечной гаммы.

# 3 Теоретическое введение

Из всех схем шифрования простейшей и наиболее надежной является схема однократного использования. Формируется -разрядная случайная двоичная последовательность - ключ шифра. Отправитель производит побитовое сложение по модулю два ( 2) ключа

и -разрядной двоичной последовательности

соответствующей посылаемому сообщению

*Гаммирование* - процедура наложения при помощи некоторой функции на исходный текст *гаммы* шифра, т.е. *псевдослучайной последователыюсти (ПСП)* с выходов генератора (**сокол2019шифрование?**). Псевдослучайная последовательность по своим статистическим свойствам неотличима от случайной последовательности, но является детерминированной, т.е. известен алгоритм ее формирования. Чаще Обычно в качестве функции берется операция поразрядного сложения по модулю два или по модулю ( - число букв алфавита открытого текста).

Простейший генератор псевдослучайной последовательности можно представить рекуррентным соотношением:

$$
\gamma\_i = a\cdot\gamma\_{i-1} + b\:\mathnormal{mod(m)}, i = \overline{1,m}
$$

где - -й член последовательности псевдослучайных чисело, - ключевые параметры.

При использовании генератора ПСП получаем бесконечную гамму. Однако, возможен режим шифрования конечной гаммы. В роли конечной гаммы может выступать фраза. Как и ранее, используется алфавитный порядок букв.

**Замечание**

В примере, данном в описании лабораторной работы, допущена ошибка - берется алфавит без буквы “**ё**”, т.е. алфавит длины 32, хотя указан алфавит длины 33.

# 4 Выполнение лабораторной работы

Для реализации шифров мы будем использовать Python, так как его синтаксис позволяет быстро реализовать необходимые нам алгоритмы.

## 4.1 Модули и вспомогательные фукнции

Дополнительно мы используем библиотеку numpy и импортируем её.

import numpy as np

Также, реализовали функцию получения английского и русского алфавита.

# Cyrillic or Latin alphabet getter  
def get\_alphabet(option="eng"):  
 if option == "eng":  
 return list(map(chr, range(ord("a"), ord("z")+1)))  
 elif option == "rus":  
 return list(map(chr, range(ord("а"), ord("я")+1)))

## 4.2 Реализация шифрования гаммированием

Шифрование гаммированием реализуем в виде функции gamma\_encryption следующего вида:

# Gamma Encryption  
 def gamma\_encryption(message: str, gamma: str):  
 alphabet = get\_alphabet()  
 if message.lower() not in alphabet:  
 alphabet = get\_alphabet("rus")  
   
 print(alphabet)  
 m = len(alphabet)  
  
 def encrypt(letters\_pair: tuple):  
 idx = (letters\_pair[0] + 1) + (letters\_pair[1] + 1) % m  
 if idx > m:  
 idx = idx - m  
  
 return idx - 1  
   
 message\_cleared = list(filter(lambda s: s.lower() in alphabet, message))  
 gamma\_cleared = list(filter(lambda s: s.lower() in alphabet, gamma))  
   
 message\_indices = list(map(lambda s: alphabet.index(s.lower()), message\_cleared))  
 gamma\_indices = list(map(lambda s: alphabet.index(s.lower()), gamma\_cleared))  
   
 for i in range(len(message\_indices) - len(gamma\_indices)):  
 gamma\_indices.append(gamma\_indices[i])  
  
 print(f'{message.upper()} -> {message\_indices}\n{gamma.upper()} -> {gamma\_indices}')  
  
 encrypted\_indices = list(map(lambda s: encrypt(s), zip(message\_indices, gamma\_indices)))  
 print(f"ENCRYPTED FORM: {encrypted\_indices}\n")  
   
 return ''.join(list(map(lambda s: alphabet[s], encrypted\_indices))).upper()

На вход она принимает переменные message (передаваемое сообщение), gamma (гамма-ключ).

В ходе обработке мы работаем с индексами элементов массива-строки, предварительно проверяя, является ли первый элемент сообщения частью передаваемого алфавита.

Далее, мы очищаем сообщений от символов, не входящих в алфавит, что дает нам возможность сделать соответствие индексов гаммы-ключа и самого сообщения.

В результате, каждый из символов проходит процедуру шифрованию, и мы получаем зашифрованное сообщение по определенной гамме.

## 4.3 Тестирование

Для тестирования мы создали следующую функцию, которую вызываем в блоке *Main*:

# --- Tests ---  
def test\_encryption(message: str, gamma: str):  
 print(f'ENCRYPTION RESULT: {gamma\_encryption(message, gamma)}')

Данные тесты возвращают строку шифро-текста в качестве результата.

Для их вызова, реализуем функцию main следующим образом:

# --- Main function ---  
def main():  
 message = "приказ"  
 gamma = "гамма"  
  
 print("TEST 1\n")  
 test\_encryption(message, gamma)  
  
 message = "лайк подписка колокольчик"  
 gamma = "нижний текст"  
  
 print("TEST 2\n")  
 test\_encryption(message, gamma)

## 4.4 Результаты тестирования

Запустив наш программный код, получим вот такой результат:

TEST 1  
  
['а', 'б', 'в', 'г', 'д', 'е', 'ж', 'з', 'и', 'й', 'к', 'л', 'м', 'н', 'о', 'п', 'р', 'с', 'т', 'у', 'ф', 'х', 'ц', 'ч', 'ш', 'щ', 'ъ', 'ы', 'ь', 'э', 'ю', 'я']  
ПРИКАЗ -> [15, 16, 8, 10, 0, 7]  
ГАММА -> [3, 0, 12, 12, 0, 3]  
ENCRYPTED FORM: [19, 17, 21, 23, 1, 11]  
  
ENCRYPTION RESULT: УСХЧБЛ  
TEST 2  
  
['а', 'б', 'в', 'г', 'д', 'е', 'ж', 'з', 'и', 'й', 'к', 'л', 'м', 'н', 'о', 'п', 'р', 'с', 'т', 'у', 'ф', 'х', 'ц', 'ч', 'ш', 'щ', 'ъ', 'ы', 'ь', 'э', 'ю', 'я']  
ЛАЙК ПОДПИСКА КОЛОКОЛЬЧИК -> [11, 0, 9, 10, 15, 14, 4, 15, 8, 17, 10, 0, 10, 14, 11, 14, 10, 14, 11, 28, 23, 8, 10]  
НИЖНИЙ ТЕКСТ -> [13, 8, 6, 13, 8, 9, 18, 5, 10, 17, 18, 13, 8, 6, 13, 8, 9, 18, 5, 10, 17, 18, 13]  
ENCRYPTED FORM: [25, 9, 16, 24, 24, 24, 23, 21, 19, 3, 29, 14, 19, 21, 25, 23, 20, 1, 17, 7, 9, 27, 24]  
  
ENCRYPTION RESULT: ЩЙРШШШЧХУГЭОУХЩЧФБСЗЙЫШ

Сравнивая результат шифрования с примером из описания лабораторной работы, можем убедиться, что наша реализация корректна.

# 5 Выводы

В рамках выполненной лабораторной работы мы изучили и реализовали алгоритм шифрования гаммированием конечной гаммы.

# Список литературы