Отчет по лабораторной работе №3

Шифрование гаммированием

Бурдина Ксения Павловна

13 октября 2023

Содержание

Список иллюстраций

1. Цель работы

Целью данной работы является освоение шифрования гаммированием, а также его программная реализация.

2. Задание

1. Изучить способ шифрования гаммированием.
2. Реализовать алгоритм шифрования гаммированием конечной гаммой.

3. Теоретическое введение

Из всех схем шифрования простейшей и наиболее надежной является схема однократного использования:

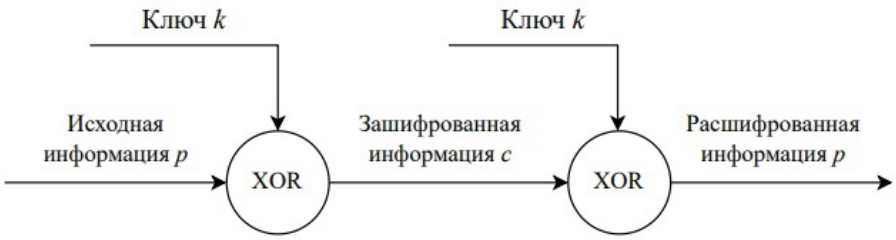

Рис. 3.1.: Схема однократного использования

Формируется $m-$ разрядная случайная двоичная последовательность - ключ шифра. Отправитель производит побитовое сложение по модулю два ($mod2$) ключа $k = k_1k_2...k_i...k_m$ и $m-$ разрядной двоичной последовательности $p = p_1p_2...p_i...p_m$, соответствующей посылаемому сообщению:

$$c_i = p_i \oplus k_i, i = \overline{1, m}$$

где p_i - $i-$й бит исходного текста, k_i - $i-$й бит ключа, $\oplus$ - операция побитового сложения (XOR), c_i - $i-$й бит получившейся криптограммы: $c = c_1c_2...c_i...c_m$.

Операция побитного сложения является обратимой, то есть $(x \oplus y) \oplus y = x$, поэтому дешифрование осуществляется повторным применением операции $\oplus$ к криптограмме:

$$p_i = c_i \oplus k_i, i = \overline{1, m}$$

Основным недостатком такой схемы является равенство объема ключевой информации и суммарного объема передаваемых сообщений. Данный недостаток можно убрать, использовав ключ в качестве “зародыша”, порождающего значительно более длинную ключевую последовательность.

На данном рисунке представлена такая схема, которая и называется гаммированием:

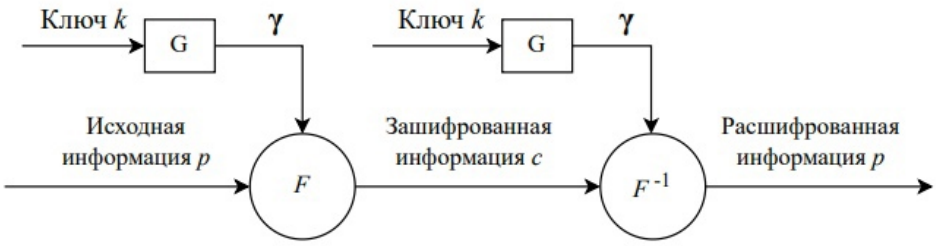

Рис. 3.2.: Шифрование гаммированием

Гаммирование - процедура наложения при помощи некоторой функции F на исходный текст гаммы шифра, то есть псевдослучайной последовательности (ПСП) с выходом генератора G. Псевдослучайная последовательность по своим статистическим свойствам неотличима от случайной последовательности, но является детерминированной, то есть известен алгоритм ее формирования. Обычно в качестве функции F берется операция поразрядного сложения по модулю два или по модулю N (N - число букв алфавита открытого текста) [1].

Простейший генератор псевдослуайной последовательности можно представить рекуррентным соотношением:

$$\gamma_i = a * \gamma_{i-1} + b * mod(m), i = \overline{1, m}$$

где γ_i - i-й член последовательности псевдослучайных чисел, a, γ_0, b - ключевые параметры. Такая последовательность состоит из целых чисел от 0 до m-1. Если элементы γ_i и γ_j совпадут, то совпадут и последующие участки: $\gamma_{i+1} = \gamma_{j+1}, \gamma_{i+2} = \gamma_{j+2}$. Таким образом, ПСП является периодической. Знание пери-

ода гаммы существенно облегчает криптоанализ. Максимальная длина периода равна m. Для ее достижения необходимо удовлетворить следующим условиям:

- b и m - взаимно простые числа;
- a-1 делится на любой простой делитель числа m;
- a-1 кратно 4, если m кратно 4.

Стойкость шифров, основанных на процедуре гаммрования, зависит от характеристик гаммы - длины и равномерности распределения вероятностей появления знаков гаммы [2].

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, то есть буква “а” имеет порядквый номер 1, “б” - 2 и так далее.

Например, зашифруем слово “ПРИКАЗ” (“16 17 09 11 01 08”) гаммой “ГАММА” (“04 01 13 13 01”). Будем использовать операцию побитового сложения по модулю 33 (mod 33). Получаем криптограмму: “УСХЧБЛ” (“20 18 22 24 02 12”).

4. Ход выполнения лабораторной работы

Для реализации шифров перестановки будем использовать среду JupyterLab. Выполним необходимую задачу.

1. Зададим функцию определения алфавита для дальнейшей работы с шифрами перестановки:

```
import numpy as np
```

```
def get_alph(option):
    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))))
```

Рис. 4.1.: Алфавит для реализации шифров

2. Пропишем функцию, в которой запишем принцип формирования алгоритма конечной гаммой:

```python
def gamma_encrypt(message: str, gamma: str):
    alph = get_alph('eng')
    if message.lower() not in alph:
        alph = get_alph('rus')
    print(alph)
    m = len(alph)
    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_clear = list(filter(lambda s: s.lower() in alph, message))
    gamma_clear = list(filter(lambda s: s.lower() in alph, gamma))
    message_ind = list(map(lambda s: alph.index(s.lower()), message_clear))
    gamma_ind = list(map(lambda s: alph.index(s.lower()), gamma_clear))
    for i in range(len(message_ind) - len(gamma_ind)):
        gamma_ind.append(gamma_ind[i])
    print(f'{message.upper()} -> {message_ind}\n{gamma.upper()} -> {gamma_ind}')
    encrypted_ind = list(map(lambda s: encrypt(s), zip(message_ind, gamma_ind)))
    print(f'encrypted form: {encrypted_ind}\n')
    return ''.join(list(map(lambda s: alph[s], encrypted_ind))).upper()
```

Рис. 4.2.: Функция алгоритма шифрования конечной гаммой

Здесь мы подаем на ввод некоторый текст, определяем его алфавит и выводим на экран. Далее определяем длину исходного текста. Задаем функцию для нахождения индексов каждого символа в заданном сообщении. После этого работаем с командами ввода сообщения и преобразования его с помощью конечной гаммы, находим индексы символов заданной гаммы и накладываем шифр заданной гаммой поверх исходного текста.

Далее формируем зашифрованный текст с помощью схемы гаммирования, то есть складываем индексы побитно и получаем индексы для шифровки. Выводим на экран полученный шифр в виде символов, а также их индексацию.

3. Создаем функцию для принятия вводимого текста и его преобразования. Для этого передаем в функцию значения исходного текста и гаммы, которая будет накладываться на текст. После этого вводим проверочные данные и вызываем функцию для работы алгоритма:

```
def test_encryption(message: str, gamma: str):
    print(f'encryption result: {gamma_encrypt(message, gamma)}')
```

```
message = 'ПРИКАЗ'
gamma = 'ГАММА'
test_encryption(message, gamma)
```

```
['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
```

Рис. 4.3.: Реализация шифрования гаммированием на примере

Видим, что полученное зашифрованное сообщение получилось аналогичным с примером, соответственно, шифрование выполнено верно.

4. Для проверки корректной работы алгоритма введем еще несколько данных:

```
message = 'ЗДЕСЬ МОГЛА БЫ БЫТЬ ВАША РЕКЛАМА'
gamma = 'КОЛИЗЕЙ'
test_encryption(message, gamma)
```

```
['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']
ЗДЕСЬ МОГЛА БЫ БЫТЬ ВАША РЕКЛАМА -> [7, 4, 5, 17, 28, 12, 14, 3, 11, 0, 1, 27,
1, 27, 18, 28, 2, 0, 24, 0, 16, 5, 10, 11, 0, 12, 0]
КОЛИЗЕЙ -> [10, 14, 11, 8, 7, 5, 9, 10, 14, 11, 8, 7, 5, 9, 10, 14, 11, 8, 7,
5, 9, 10, 14, 11, 8, 7, 5]
encrypted form: [18, 19, 17, 26, 4, 18, 24, 14, 26, 12, 10, 3, 7, 5, 29, 11, 1
4, 9, 0, 6, 26, 16, 25, 23, 9, 20, 6]

encryption result: ТУСЪДТШОЪМКГЗЕЭЛОЙАЖЪРЩЧЙФЖ
```

Рис. 4.4.: Результат шифрования гаммированием на русском

```
message = 'FEEL THE RUSSIAN STYLE'
gamma = 'HATTERS'
test_encryption(message, gamma)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
FEEL THE RUSSIAN STYLE -> [5, 4, 4, 11, 19, 7, 4, 17, 20, 18, 18, 8, 0, 13, 18,
19, 24, 11, 4]
HATTERS -> [7, 0, 19, 19, 4, 17, 18, 7, 0, 19, 19, 4, 17, 18, 7, 0, 19, 19, 4]
encrypted form: [13, 5, 24, 5, 24, 25, 23, 25, 21, 12, 12, 13, 18, 6, 0, 20, 1
8, 5, 9]

encryption result: NFYFYZXZVMMNSGAUSFJ
```

Рис. 4.5.: Результат шифрования гаммированием на английском

Если посмотреть по индексам символов, видно, что алгоритм работает корректно, значит, шифрование гаммированием конечной гаммой реализовано верно.

5. Листинг программы

```
import numpy as np

def get_alph(option):
  if option=='eng':
    return(list(map(chr, range(ord('a'), ord('z')+1))))
  elif option=='rus':
    return(list(map(chr, range(ord('a'), ord('я')+1))))

def gamma_encrypt(message: str, gamma: str):
  alph = get_alph('eng')
  if message.lower() not in alph:
    alph = get_alph('rus')
  print(alph)
  m = len(alph)
  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_clear = list(filter(lambda s: s.lower() in alph, message))
  gamma_clear = list(filter(lambda s: s.lower() in alph, gamma))
  message_ind = list(map(lambda s: alph.index(s.lower()), message_clear))
```

```
  gamma_ind = list(map(lambda s: alph.index(s.lower()), gamma_clear))
  for i in range(len(message_ind) - len(gamma_ind)):
    gamma_ind.append(gamma_ind[i])
  print(f'{message.upper()} -> {message_ind}\n{gamma.upper()} -> {gamma_ind}')
  encrypted_ind = list(map(lambda s: encrypt(s), zip(message_ind, gamma_ind)))
  print(f'encrypted form: {encrypted_ind}\n')
  return ''.join(list(map(lambda s: alph[s], encrypted_ind))).upper()

def test_encryption(message: str, gamma: str):
  print(f'encryption result: {gamma_encrypt(message, gamma)}')

message = 'ПРИКАЗ'
gamma = 'ГАММА'
test_encryption(message, gamma)

message = 'ЗДЕСЬ МОГЛА БЫ БЫТЬ ВАША РЕКЛАМА'
gamma = 'КОЛИЗЕЙ'
test_encryption(message, gamma)

message = 'FEEL THE RUSSIAN STYLE'
gamma = 'HATTERS'
test_encryption(message, gamma)
```

6. Выводы

В ходе работы мы изучили и реализовали алгоритм шифрования гаммированием конечной гаммой.

7. Список литературы

1. Традиционные шифры с симметричным ключом [1]

2. Методические материалы курса [2]