

# **Лабораторная работа №8**

**Элементы криптографии. Шифрование (кодирование) различных  
исходных текстов одним ключом**

Латыпова Диана

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
3.1	Введение в шифрование . . . . .	7
3.2	Основы однократного гаммирования . . . . .	7
3.3	Преимущества и недостатки . . . . .	8
3.3.1	Преимущества: . . . . .	8
3.3.2	Недостатки: . . . . .	8
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>14</b>
5.0.1	1. Как, зная один из текстов (P1 или P2), определить другой, не зная при этом ключа? . . . . .	14
5.0.2	2. Что будет при повторном использовании ключа при шифровании текста? . . . . .	14
5.0.3	3. Как реализуется режим шифрования однократного гаммирования одним ключом двух открытых текстов? . . . . .	15
5.0.4	4. Перечислите недостатки шифрования одним ключом двух открытых текстов. . . . .	15
5.0.5	5. Перечислите преимущества шифрования одним ключом двух открытых текстов. . . . .	15
<b>6</b>	<b>Выводы</b>	<b>17</b>
	<b>Список литературы</b>	<b>18</b>

## Список иллюстраций

4.1	Вызов функции <code>hask</code> . . . . .	10
4.2	Шифрование/расшифрование текста . . . . .	13

## **Список таблиц**

# 1 Цель работы

Освоить 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  $P1$  и  $P2$  в режиме однократного гаммирования. Приложение должно определить вид шифротекстов  $C1$  и  $C2$  обоих текстов  $P1$  и  $P2$  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 Введение в шифрование

Шифрование является важной частью информационной безопасности, обеспечивая конфиденциальность данных путем их преобразования в неразборчивый вид. Одним из наиболее простых и при этом эффективных методов шифрования является **однократное гаммирование** (или шифрование с использованием одно-разового блокнота), которое обеспечивает идеальную стойкость при правильном использовании[1].

### 3.2 Основы однократного гаммирования

**Однократное гаммирование** — это метод шифрования, при котором каждый бит открытого текста комбинируется с битом ключа [2] с помощью операции *исключающего ИЛИ* (XOR) [3]. Этот метод основан на следующей формуле:

$$[ C = P \oplus K ]$$

где: - ( C ) — шифртекст, - ( P ) — открытый текст, - ( K ) — ключ.

Ключ должен быть случайным, иметь ту же длину, что и открытый текст, и использоваться только один раз. Это условие обеспечивает идеальную стойкость метода: даже если злоумышленник получит доступ к шифртексту, он не сможет восстановить открытый текст без знания ключа.

## 3.3 Преимущества и недостатки

### 3.3.1 Преимущества:

- **Идеальная стойкость:** Однократное гаммирование предоставляет идеальную стойкость, если ключ является случайным и не используется повторно.
- **Простота реализации:** Метод легко реализуется, так как XOR является простой и быстрой операцией.
- **Гибкость:** Однократное гаммирование может использоваться для шифрования данных любой длины, при условии, что ключ такой же длины.

### 3.3.2 Недостатки:

- **Проблемы с управлением ключами:** Для каждого нового сообщения требуется уникальный ключ, что может усложнить процесс хранения и передачи ключей.
- **Уязвимость к атаке при повторном использовании ключа:** Если один и тот же ключ используется для шифрования нескольких сообщений, это может раскрыть информацию о открытых текстах.
- **Необходимость в хранении ключей:** Долговременное хранение и безопасная передача ключей могут представлять собой сложность.



## 4 Выполнение лабораторной работы

Первая часть кода:

```
# Создаем алфавит из русских букв и цифр для гаммирования
start_char = ord("a")
alphabeth = [chr(i) for i in range(start_char, start_char + 32)] # Добавляем строчные
start_char = ord("0")
for i in range(start_char, start_char + 10):
    alphabeth.append(chr(i)) # Добавляем цифры (0-9)

start_char = ord("А")
for i in range(1040, 1072):
    alphabeth.append(chr(i)) # Добавляем заглавные буквы русского алфавита (32 буквы)
print(alphabeth)

# Измененные тексты для шифрования
P1 = "НаВашисходящийот1204"
P2 = "ВСеверныйфилиалБанка"

# Задан ключ длиной 20 символов для гаммирования
K = "05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54"

# Функция взлома, которая ищет закономерности между двумя зашифрованными текстами P1 и P2
def hack(P1, P2):
```

```

xor_code = []

# Для каждого символа выполняется сложение индексов символов из двух текстов
for i in range(20):
    xor_code.append(alphabeth[(alphabeth.index(P1[i]) + alphabeth.index(P2[i])) % len(alphabeth)])

# Выводим результат сложения символов для визуального анализа
print(xor_code)

print(xor_code[16], " и ", xor_code[19]) # Показываем определенные символы для анализа
combined_text = "".join(xor_code) # Собираем итоговый текст из символов
print(combined_text)

hack(P1, P2) # Вызываем функцию для взлома

```

Вызвали функцию `hack` (рис. 4.1).

```

P1 = "НаВашисходящий1204"
P2 = "ВСеве́рныйфилиалБанка"
K = "05 0C 17 7F 0E 4E 37 D2 94 10 09 2E 22 57 FF C8 0B B2 70 54"

def hack(P1, P2):
    xor_code = []
    for i in range(20):
        xor_code.append(alphabeth[(alphabeth.index(P1[i]) + alphabeth.index(P2[i])) % len(alphabeth)])
    print(xor_code)
    print(xor_code[16], " и ", xor_code[19])
    combined_text = "".join(xor_code)
    print(combined_text)

hack(P1, P2)

```

['щ', 'с', 'з', 'в', 'э', 'ш', 'ю', 'ж', 'ч', 'ш', '7', '4', 'р', 'й', 'щ', 'у', '1', 'Е', 'А', '4']  
1 и 4  
щсзвэшжчш74рйщу1ЕА4

Рис. 4.1: Вызов функции `hack`

Код для шифрования и расшифрования текста:

```

# Функция для шифрования и дешифрования текста с использованием метода однократного га
def encrypt(P1):
    # Создаем словарь, который связывает символы русского алфавита и цифры с числами
    char_to_num = {"а": 1, "б": 2, "в": 3, "г": 4, "д": 5, "е": 6, "ё": 7, "ж": 8, "з": 9, "и": 10, "к": 11, "л": 12, "м": 13, "н": 14, "о": 15, "п": 16, "р": 17, "с": 18, "т": 19, "у": 20, "ф": 21, "х": 22, "ц": 23, "ш": 24, "щ": 25, "ъ": 26, "ы": 27, "ь": 28, "э": 29, "ю": 30, "я": 31, "А": 32, "Б": 33, "В": 34, "Г": 35, "Д": 36, "Е": 37, "Ё": 38, "Ж": 39, "З": 40, "И": 41, "Й": 42, "К": 43, "Л": 44, "М": 45, "Н": 46, "О": 47, "П": 48, "Р": 49, "С": 50, "Т": 51, "У": 52, "Ф": 53, "Х": 54, "Ц": 55, "Ш": 56, "Щ": 57, "Ъ": 58, "Ы": 59, "Ь": 60, "Э": 61, "Ю": 62, "Я": 63}

```

```
"П": 49, "Р": 50, "С": 51, "Т": 52, "У": 53, "Ф": 54, "Х": 55, "Ц":  
"Ы": 61, "Ь": 62, "Э": 63, "Ю": 64, "Я": 65, "1": 66, "2": 67, "3":  
"8": 73, "9": 74, "0": 75}
```

```
# Создаем обратный словарь для декодирования чисел в символы
```

```
num_to_char = {v: k for k, v in char_to_num.items()}
```

```
input_text = P1 # Входной текст
```

```
gamma = input("Введите гамму (только символы из словаря): ") # Запрашиваем у поль
```

```
text_nums = []
```

```
gamma_nums = []
```

```
# Преобразуем текст и гамму в числовые значения
```

```
for char in input_text:
```

```
    text_nums.append(char_to_num[char])
```

```
print("Числа текста", text_nums) # Показываем числовое представление текста
```

```
for char in gamma:
```

```
    gamma_nums.append(char_to_num[char])
```

```
print("Числа гаммы", gamma_nums) # Показываем числовое представление гаммы
```

```
encrypted_nums = []
```

```
idx = 0
```

```
# Шифрование текста путем сложения чисел текста и гаммы
```

```
for char in input_text:
```

```
    try:
```

```
        new_num = char_to_num[char] + gamma_nums[idx] # Сложение числа символа те
```

```

except:
    idx = 0 # Если индекс гаммы выходит за пределы, обнуляем его
    new_num = char_to_num[char] + gamma_nums[idx]

    if new_num > 75: # Если число больше 75 (максимальный индекс), делаем модуль
        new_num = new_num % 75
    idx += 1
    encrypted_nums.append(new_num) # Добавляем зашифрованное число в список

print("Числа зашифрованного текста", encrypted_nums)

encrypted_text = ""
for num in encrypted_nums:
    encrypted_text += num_to_char[num] # Преобразуем зашифрованные числа обратно

print("Зашифрованный текст: ", encrypted_text)

# Расшифровка текста
decrypted_nums = []
for char in encrypted_text:
    decrypted_nums.append(char_to_num[char])

idx = 0
decrypted_result = []

# Дешифрование текста путем вычитания чисел гаммы из чисел зашифрованного текста
for num in decrypted_nums:
    try:
        new_num = num - gamma_nums[idx] # Вычитаем значение гаммы

```

```

except:
    idx = 0
    new_num = num - gamma_nums[idx]

if new_num < 1: # Если число меньше 1, добавляем 75
    new_num = 75 + new_num
decrypted_result.append(new_num)
idx += 1

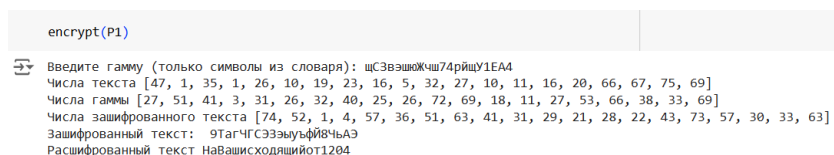
decrypted_text = ""
for num in decrypted_result:
    decrypted_text += num_to_char[num] # Преобразуем числа обратно в текст

print("Расшифрованный текст", decrypted_text) # Выводим расшифрованный текст

encrypt(P1) # Шифруем и расшифровываем текст

```

Выполнение кода(рис. 4.2).



```

encrypt(P1)
Введите гамму (только символы из словаря): шСЗвэшоЖш74рйщУ1ЕА4
Числа текста [47, 1, 35, 1, 26, 10, 19, 23, 16, 5, 32, 27, 10, 11, 16, 20, 66, 67, 75, 69]
Числа гаммы [27, 51, 41, 3, 31, 26, 32, 40, 25, 26, 72, 69, 18, 11, 27, 53, 66, 38, 33, 69]
Числа зашифрованного текста [74, 52, 1, 4, 57, 36, 51, 63, 41, 31, 29, 21, 28, 22, 43, 73, 57, 30, 33, 63]
Зашифрованный текст: 9ТагЧГСЗЗыуфЙ8ЧьАЭ
Расшифрованный текст: НаВашисходящийот1204

```

Рис. 4.2: Шифрование/расшифрование текста

## 5 Контрольные вопросы

### 5.0.1 1. Как, зная один из текстов (P1 или P2), определить другой, не зная при этом ключа?

Зная один из текстов (например, ( P1 )) и используя тот факт, что шифрование выполняется с помощью операции XOR (как это происходит в однократном гаммировании), можно вычислить другой текст ( P2 ) следующим образом: - Если известен шифртекст ( C ) и один из открытых текстов ( P1 ), то другой открытый текст можно восстановить с помощью обратного применения XOR:  $( P2 = P1 \oplus C )$ , где  $( \oplus )$  — операция побитового исключающего ИЛИ (XOR).

Пример: - Если  $( C = P1 \oplus K )$ , а  $( P2 = P1 \oplus K )$ , то зная ( P1 ), можно вычислить ( P2 ) с помощью операции  $( P2 = P1 \oplus C )$ , где ( K ) — ключ.

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

Повторное использование ключа в шифровании однократным гаммированием (одноразовым блокнотом) приводит к серьезным криптографическим уязвимостям: - Если два текста ( P1 ) и ( P2 ) зашифрованы одним и тем же ключом ( K ), то разность их шифртекстов может дать разность исходных текстов:  $( C1 \oplus C2 = P1 \oplus P2 )$ . - Это позволяет криптоаналитику вычислить некоторые характеристики исходных текстов (или даже восстановить их полностью), что делает шифрование уязвимым к атакам, таким как атака известной разности текстов.

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

Режим шифрования однократного гаммирования двумя открытыми текстами реализуется следующим образом: 1. Генерируется случайный ключ длиной, равной длине каждого из текстов ( P1 ) и ( P2 ). 2. Оба текста шифруются с использованием ключа по правилу: - ( C1 = P1  $\oplus$  K ) - ( C2 = P2  $\oplus$  K ) 3. Шифртексты ( C1 ) и ( C2 ) передаются или хранятся вместе с ключом для дальнейшей расшифровки.

### 5.0.4 4. Перечислите недостатки шифрования одним ключом двух открытых текстов.

- **Повторное использование ключа:** Как уже упоминалось, если один и тот же ключ используется для шифрования двух разных сообщений, можно вычислить разность между открытыми текстами, что может привести к раскрытию оригинальных текстов.
- **Уязвимость к атаке известной разности текстов:** Повторное использование ключа может раскрыть информацию об исходных данных, что существенно снижает стойкость шифра.
- **Проблемы с управлением ключами:** Для каждого нового сеанса шифрования требуется новый уникальный ключ, который нужно безопасно передавать и хранить. Если ключ повторно используется или его длина меньше длины сообщения, шифр теряет свою идеальную стойкость.

### 5.0.5 5. Перечислите преимущества шифрования одним ключом двух открытых текстов.

- **Простота реализации:** Процедура однократного гаммирования очень проста в реализации, так как использует операцию XOR, которая быстро выполняется.

- **Идеальная стойкость при соблюдении условий:** Однократное гаммирование является абсолютно стойким методом шифрования (невозможно вскрыть зашифрованный текст без знания ключа) при условии, что ключ используется только один раз и имеет ту же длину, что и исходный текст.
- **Отсутствие зависимости от длины текста:** Метод может применяться к сообщениям любой длины при наличии ключа той же длины, что делает его универсальным для различной информации.

Недостатки метода, связанные с ключевым управлением и повторным использованием ключей, обычно перевешивают эти преимущества, что ограничивает его использование в практических сценариях.



## 6 Выводы

Я освоила на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом.

## Список литературы

1. Шифрование [Электронный ресурс]. Wikipedia®, 2024. URL: <https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5>.
2. Однократное гаммирование. [Электронный ресурс]. Studfile, 2013. URL: <https://studfile.net/preview/272674/page:7/>.
3. Лекция 3: Простейшие методы шифрования с закрытым ключом [Электронный ресурс]. ИНТУИТ. Национальный открытый университет, 2024. URL: <https://intuit.ru/studies/courses/691/547/lecture/12373?page=4#:~:text=%D0%95%D1%89%D0%B5%20%D0%BE%D0%B4%D0%BD%D0%B8%D0%BC%20%D1%87%D0%B0%D1%81%D1%82%D0%BD%D1%8B%D0%BC%20%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B5%D0%BC%20%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82%D0%BD%D0%BE%D0%B9%20%D0%BF%D0%BE%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B8%20%D1%8F%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F%20%D0%B3%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%B0%2C%20%D1%82%D0%BE%20%D1%81%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%BF%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D1%81%D1%8F%20%D0%BF%D0%BE%20%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E%2033>.