Лабораторная работа №8

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

Латыпова Диана

Содержание

# 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, прочитать оба текста. Необходимо разработать приложение, позволяющее шифровать и дешифровать тексты и в режиме однократного гаммирования. Приложение должно определить вид шифротекстов и обоих текстов и при известном ключе ; Необходимо определить и выразить аналитически способ, при котором злоумышленник может прочитать 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 одноразового 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 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("а")  
alphabeth = [chr(i) for i in range(start\_char, start\_char + 32)] # Добавляем строчные буквы русского алфавита (32 буквы)  
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. 1).

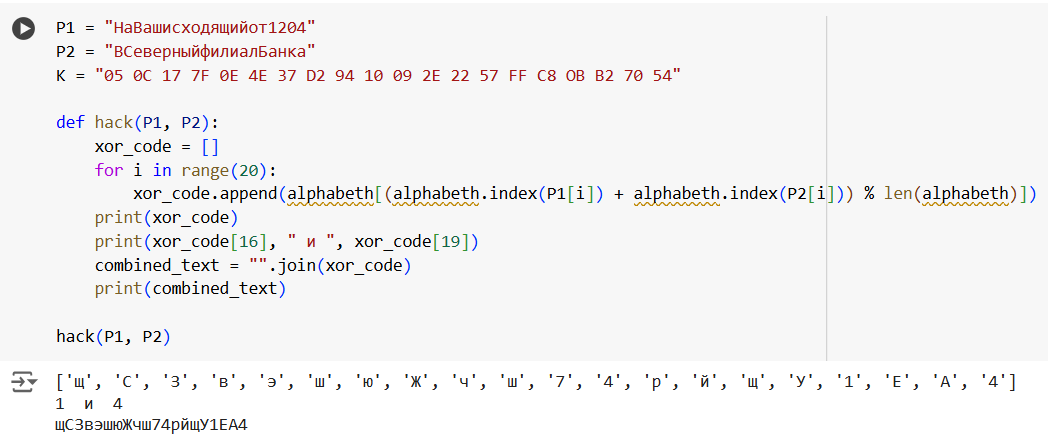


Рис. 1: Вызов функции hack

Код для шифрования и расшифрования текста:

# Функция для шифрования и дешифрования текста с использованием метода однократного гаммирования  
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, "й": 11, "к": 12, "л": 13,  
 "м": 14, "н": 15, "о": 16, "п": 17, "р": 18, "с": 19, "т": 20, "у": 21, "ф": 22, "х": 23, "ц": 24, "ч": 25,   
 "ш": 26, "щ": 27, "ъ": 28, "ы": 29, "ь": 30, "э": 31, "ю": 32, "я": 32, "А": 33, "Б": 34, "В": 35, "Г": 36,   
 "Д": 37, "Е": 38, "Ё": 39, "Ж": 40, "З": 41, "И": 42, "Й": 43, "К": 44, "Л": 45, "М": 46, "Н": 47, "О": 48,   
 "П": 49, "Р": 50, "С": 51, "Т": 52, "У": 53, "Ф": 54, "Х": 55, "Ц": 56, "Ч": 57, "Ш": 58, "Щ": 59, "Ъ": 60,   
 "Ы": 61, "Ь": 62, "Э": 63, "Ю": 64, "Я": 65, "1": 66, "2": 67, "3": 68, "4": 69, "5": 70, "6": 71, "7": 72,   
 "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 пользователя гамму  
  
 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 текста и гаммы  
 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 по 75  
 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 в текст  
  
 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. 2).

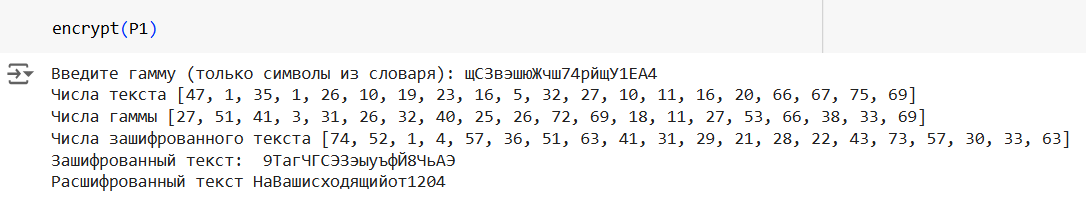


Рис. 2: Шифрование/расшифрование текста

# 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 C ), где ( ) — операция побитового исключающего ИЛИ (XOR).

Пример: - Если ( C = P1 K ), а ( P2 = P1 K ), то зная ( P1 ), можно вычислить ( P2 ) с помощью операции ( P2 = P1 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 C2 = P1 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 K ) - ( C2 = P2 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.,33%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.>