

**Анализ и применение аддитивного метода гаммирования для
шифрования текстовых данных: реализация, проверка правильности
ключа и оценка криптографической стойкости**

Екимова Яна Сергеевна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной статье представлен алгоритм шифрования и дешифрования данных методом гаммирования с использованием аддитивного датчика. Разработанный алгоритм включает функции генерации псевдослучайной последовательности чисел (гаммы), шифрования и дешифрования текста. Шифрование осуществляется путём побитового сложения каждого байта текста с соответствующим значением гаммы по модулю 256, что обеспечивает высокий уровень криптографической защиты. Для дешифрования используется обратная операция — вычитание значений гаммы из зашифрованных данных. В статье также описан механизм проверки пароля, который обеспечивает контроль доступа к процессу шифрования и дешифрования.

Ключевые слова: шифрование, дешифрование, python, google coollaboratory.

**Analysis and application of the additive gamming method for text data
encryption: implementation, key validation, and cryptographic strength
assessment**

Ekimova Yana Sergeevna

Sholom Aleichem Priamurskiy State University

Student

Abstract

This article presents an algorithm for data encryption and decryption using the gamma method using an additive sensor. The developed algorithm includes functions for generating a pseudorandom sequence of numbers (gamma), text encryption and decryption. Encryption is performed by bitwise adding each byte of the text with the corresponding gamma value modulo 256, which provides a high level of cryptographic protection. The reverse operation is used for decryption — subtraction of gamma values from encrypted data. The article also describes a password verification mechanism that provides access control to the encryption and decryption process.

Keywords: encryption, decryption, python, google coollaboratory.

1 Введение

1.1 Актуальность

Алгоритм Эль Гамала является одним из фундаментальных методов асимметричной криптографии, основанной на сложности задачи дискретного логарифмирования. В отличие от симметричных методов, таких как гаммирование, асимметричные алгоритмы используют пару ключей: открытый для шифрования и закрытый для дешифрования. Это делает их особенно полезными для обеспечения конфиденциальности и аутентификации в системах с открытыми каналами связи.

В данной статье представлена реализация алгоритма Эль Гамала, включая генерацию параметров, шифрование и дешифрование сообщений. Для работы алгоритма используются математические функции, такие как вычисление НОД, проверка числа на простоту и генерация случайных чисел. Код написан на языке Python и может быть использован для изучения принципов работы асимметричной криптографии.

1.2 Обзор исследований

М.Е. Алексеев рассмотрел алгоритм шифрования методом гаммирования по модулю 2 и его преимущества по сравнению с другими алгоритмами [1]. Предоставили методику устранения недостатков, присущих алгоритмам шифрования, основанным на методе гаммирования Д. Ю. Кубашев, А. Н. Леухин [2]. С.Ю. Строгалева, Е.А. Ефремов определили стойкость шифрования методом гаммирования [3]. В статье рассмотрели алгоритм шифрования гаммирование. Представили основные свойства гаммы. Произвели криптоанализ с помощью теста различных методов дешифрации. Определили основные преимущества и недостатки, влияющие на криптостойкость данного алгоритма И. Д. Бурчаловский А. А. Прохоренко [4]. П.Г. Фролов, Г.Е. Федотов, Р.Р. Нургалиев, С.А. Коноваленко, А.Н. Стадник показали программу реализации процедуры шифрования данных методом гаммирования [5].

1.3 Цель исследования

Цель исследования – выполнить анализ данных с помощью Google Colaboratory.

2 Материалы и методы

В данном исследовании используется Google Colaboratory для анализа данных.

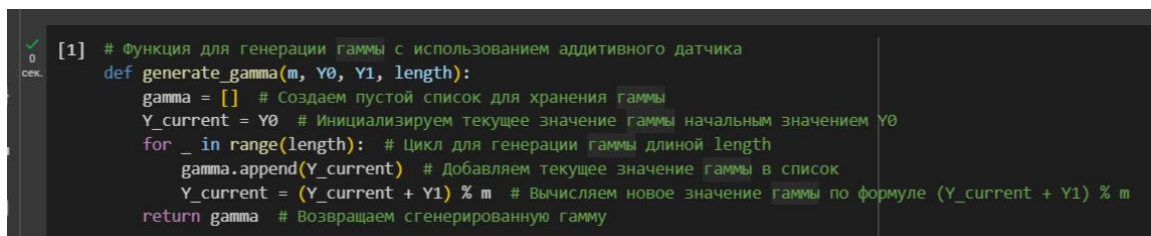
Ссылка на ноутбук:
<https://colab.research.google.com/drive/117xLRgL8KMqZE6tBvuWLzw3RMsNzvfyU?usp=sharing>.

3 Результаты

Напишем код, реализующий функцию `generate_gamma`, которая генерирует псевдослучайную последовательность чисел с использованием

аддитивного датчика начиная с начального значения Y_0 , каждое следующее значение вычисляется по формуле $(Y_{\text{current}} + Y_1) \% m$, где m — модуль, определяющий диапазон значений, а Y_1 — шаг приращения; сгенерированная последовательность длиной `length` возвращается в виде списка и может быть использована, например, для шифрования методом гаммирования.

```
# Функция для генерации гаммы с использованием аддитивного датчика
def generate_gamma(m, Y0, Y1, length):
    gamma = [] # Создаем пустой список для хранения гаммы
    Y_current = Y0 # Инициализируем текущее значение гаммы начальным значением
    Y0
    for _ in range(length): # Цикл для генерации гаммы длиной length
        gamma.append(Y_current) # Добавляем текущее значение гаммы в список
        Y_current = (Y_current + Y1) % m # Вычисляем новое значение гаммы по
        формуле (Y_current + Y1) % m
    return gamma # Возвращаем сгенерированную гамму
```

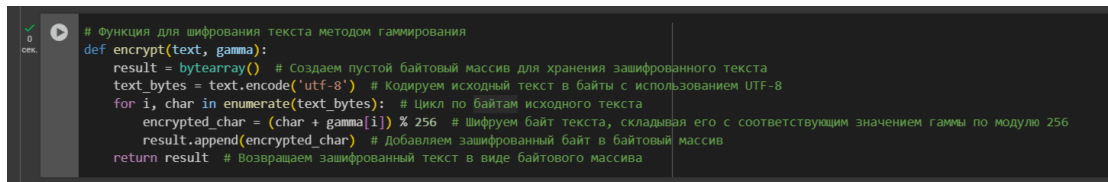


```
[1] # Функция для генерации гаммы с использованием аддитивного датчика
def generate_gamma(m, Y0, Y1, length):
    gamma = [] # Создаем пустой список для хранения гаммы
    Y_current = Y0 # Инициализируем текущее значение гаммы начальным значением Y0
    for _ in range(length): # Цикл для генерации гаммы длиной length
        gamma.append(Y_current) # Добавляем текущее значение гаммы в список
        Y_current = (Y_current + Y1) % m # Вычисляем новое значение гаммы по формуле (Y_current + Y1) % m
    return gamma # Возвращаем сгенерированную гамму
```

Рисунок 1. Код, реализующий функцию `generate_gamma`

Теперь напишем код, реализующий функцию `encrypt`, которая шифрует текст методом гаммирования: исходный текст кодируется в байты с использованием UTF-8, после чего каждый байт текста складывается с соответствующим значением гаммы по модулю 256, что обеспечивает побитовое шифрование. Результатом работы функции является байтовый массив, содержащий зашифрованный текст. Этот метод гарантирует, что зашифрованные данные остаются в диапазоне от 0 до 255, что делает их пригодными для дальнейшей обработки или передачи.

```
# Функция для шифрования текста методом гаммирования
def encrypt(text, gamma):
    result = bytearray() # Создаем пустой байтовый массив для хранения
    зашифрованного текста
    text_bytes = text.encode('utf-8') # Кодировем исходный текст в байты с
    использованием UTF-8
    for i, char in enumerate(text_bytes): # Цикл по байтам исходного текста
        encrypted_char = (char + gamma[i]) % 256 # Шифруем байт текста,
        складывая его с соответствующим значением гаммы по модулю 256
        result.append(encrypted_char) # Добавляем зашифрованный байт в
        байтовый массив
    return result # Возвращаем зашифрованный текст в виде байтового
    массива
```

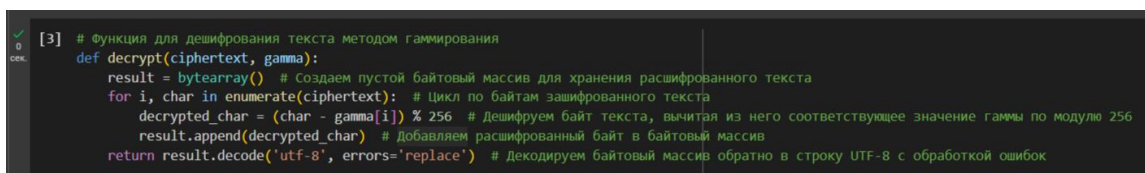


```
# Функция для шифрования текста методом гаммирования
def encrypt(text, gamma):
    result = bytearray() # Создаем пустой байтовый массив для хранения зашифрованного текста
    text_bytes = text.encode('utf-8') # Кодировем исходный текст в байты с использованием UTF-8
    for i, char in enumerate(text_bytes): # Цикл по байтам исходного текста
        encrypted_char = (char + gamma[i]) % 256 # Шифруем байт текста, складывая его с соответствующим значением гаммы по модулю 256
        result.append(encrypted_char) # Добавляем зашифрованный байт в байтовый массив
    return result # Возвращаем зашифрованный текст в виде байтового массива
```

Рисунок 2. Код, реализующий функцию encrypt

Далее напишем код, реализующий функцию decrypt, которая дешифрует текст, зашифрованный методом гаммирования: каждый байт зашифрованного текста вычитается из соответствующего значения гаммы по модулю 256, что восстанавливает исходный байт текста. Результат записывается в байтовый массив, который затем декодируется обратно в строку UTF-8. Если в процессе декодирования возникают ошибки (например, из-за некорректных данных), они заменяются специальным символом замены. Результатом работы функции является расшифрованный текст в виде строки.

```
# Функция для дешифрования текста методом гаммирования
def decrypt(ciphertext, gamma):
    result = bytearray() # Создаем пустой байтовый массив для хранения
    расшифрованного текста
    for i, char in enumerate(ciphertext): # Цикл по байтам зашифрованного текста
        decrypted_char = (char - gamma[i]) % 256 # Дешифруем байт текста, вычитая из
        него соответствующее значение гаммы по модулю 256
        result.append(decrypted_char) # Добавляем расшифрованный байт в байтовый
        массив
    return result.decode('utf-8', errors='replace') # Декодируем байтовый массив обратно
    в строку UTF-8 с обработкой ошибок
```



```
[3] # Функция для дешифрования текста методом гаммирования
def decrypt(ciphertext, gamma):
    result = bytearray() # Создаем пустой байтовый массив для хранения расшифрованного текста
    for i, char in enumerate(ciphertext): # Цикл по байтам зашифрованного текста
        decrypted_char = (char - gamma[i]) % 256 # Дешифруем байт текста, вычитая из него соответствующее значение гаммы по модулю 256
        result.append(decrypted_char) # Добавляем расшифрованный байт в байтовый массив
    return result.decode('utf-8', errors='replace') # Декодируем байтовый массив обратно в строку UTF-8 с обработкой ошибок
```

Рисунок 3. Код, реализующий функцию decrypt

Затем напишем код, который задает параметры для генерации последовательности чисел по методу линейного конгруэнтного генератора. Здесь m представляет собой модуль (в данном случае 4096), который определяет диапазон возможных значений, Y_0 — начальное значение (4003), и Y_1 — шаг (59), который используется для вычисления следующего значения в последовательности. Результатом выполнения этого кода будет последовательность чисел, генерируемая по формуле, основанной на этих параметрах, что может быть использовано для различных приложений, таких как генерация случайных чисел или симуляции.

```
# Параметры для генерации гаммы
m = 4096 # Модуль
```

$Y_0 = 4003$ # Начальное значение

$Y_1 = 59$ # Шаг

```
[4] # Параметры для генерации гаммы
m = 4096 # Модуль
Y0 = 4003 # Начальное значение
Y1 = 59 # Шаг
```

Рисунок 4. Параметры для генерации гаммы

Напишем код, который реализует механизм проверки пароля: задаётся правильный пароль `correct_password = "1711"`, после чего пользователю предлагается ввести пароль с помощью функции `input()`. Введённый пароль сохраняется в переменной `password` для последующей проверки на соответствие правильному значению. Этот механизм может быть использован для контроля доступа к ключам шифрования или другим защищённым данным.

Ввод пароля

`correct_password = "1711"` # Задаем правильный пароль

`password = input("Введите пароль для проверки ключа: ")`

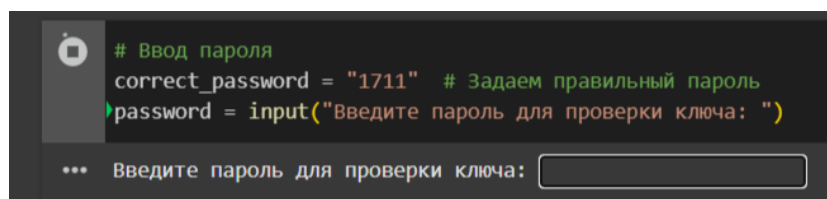


Рисунок 5. Ввод пароля

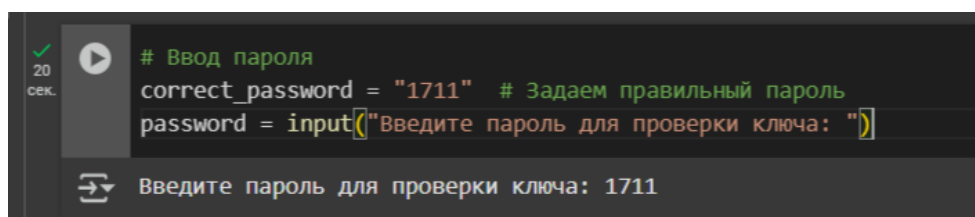


Рисунок 6. Ввод пароля

Напишем код, который выполняет проверку введенного пароля: если пароль не совпадает с заданным значением `correct_password`, выводится сообщение об ошибке и операция отменяется. В случае правильного пароля генерируется гамма для шифрования пароля с помощью функции `generate_gamma`, после чего пароль шифруется методом гаммирования и выводится в зашифрованном виде. Затем создаётся гамма для шифрования текстового сообщения, которое также шифруется и выводится. После этого выполняется дешифрование пароля и текста с использованием соответствующих гамм, и результаты выводятся на экран, демонстрируя корректность работы алгоритма шифрования и дешифрования.

```
# Проверка правильности пароля
if password != correct_password:
    print("Неправильный пароль. Операция отменена.")
else:
    # Генерация гаммы для пароля
    gamma_password = generate_gamma(m, Y0, Y1, len(password)) # Генерируем гамму
для пароля

    # Шифрование пароля
    encrypted_password = encrypt(password, gamma_password) # Шифруем пароль
    print(f"Зашифрованный пароль: {encrypted_password}") # Выводим
зашифрованный пароль

    # Пример текста для шифрования
    plaintext = "Привет, это Яна!"
    # Генерация гаммы для текста
    gamma = generate_gamma(m, Y0, Y1, len(plaintext.encode('utf-8'))) # Генерируем
гамму длиной, равной длине байтов исходного текста

    # Шифрование текста
    ciphertext = encrypt(plaintext, gamma) # Шифруем исходный текст с
использованием сгенерированной гаммы
    print(f"Зашифрованный текст: {ciphertext.decode('utf-8', errors='replace')}") #
Выводим зашифрованный текст

    # Дешифрование пароля
    decrypted_password = decrypt(encrypted_password, gamma_password) # Дешифруем
пароль
    print(f"Расшифрованный пароль: {decrypted_password}") # Выводим
расшифрованный пароль

    # Дешифрование (используем ту же гамму)
    decrypted_text = decrypt(ciphertext, gamma)
    print(f"Расшифрованный текст: {decrypted_text}") # Выводим расшифрованный
текст
```

```

# Проверка правильности пароля
if password != correct_password:
    print("Неправильный пароль. Операция отменена.")
else:
    # Генерация гаммы для пароля
    gamma_password = generate_gamma(m, Y0, Y1, len(password)) # Генерируем гамму для пароля

    # Шифрование пароля
    encrypted_password = encrypt(password, gamma_password) # Шифруем пароль
    print(f"Зашифрованный пароль: {encrypted_password}") # Выводим зашифрованный пароль

    # Пример текста для шифрования
    plaintext = "Привет, это Яна!"
    # Генерация гаммы для текста
    gamma = generate_gamma(m, Y0, Y1, len(plaintext.encode('utf-8'))) # Генерируем гамму длиной, равной длине байтов исходного текста

    # Шифрование текста
    ciphertext = encrypt(plaintext, gamma) # Шифруем исходный текст с использованием сгенерированной гаммы
    print(f"Зашифрованный текст: {ciphertext.decode('utf-8', errors='replace')}") # Выводим зашифрованный текст

    # Дешифрование пароля
    decrypted_password = decrypt(encrypted_password, gamma_password) # Дешифруем пароль
    print(f"Расшифрованный пароль: {decrypted_password}") # Выводим расшифрованный пароль

    # Дешифрование (используем ту же гамму)
    decrypted_text = decrypt(ciphertext, gamma)
    print(f"Расшифрованный текст: {decrypted_text}") # Выводим расшифрованный текст

```

Рисунок 7. Код

```

➡ Зашифрованный пароль: bytearray(b'\xd4\x15J\x85')
Зашифрованный текст: s}??_??Kk@??$E??_Jd??6Q?
Расшифрованный пароль: 1711
Расшифрованный текст: Привет, это Яна!

```

Рисунок 8. Результат алгоритма шифрования и дешифрования

Если пароль будет введён неправильно, то будет ошибка:

```

➡ Неправильный пароль. Операция отменена.

```

Рисунок 9. Ошибка

4 Выводы

В ходе работы был разработан и реализован алгоритм шифрования и дешифрования данных методом гаммирования с использованием аддитивного датчика. Функция `generate_gamma` позволяет генерировать псевдослучайную последовательность чисел, которая используется для шифрования текста. Шифрование осуществляется с помощью функции `encrypt`, где каждый байт текста складывается с соответствующим значением гаммы по модулю 256, что обеспечивает побитовое шифрование. Для дешифрования используется функция `decrypt`, которая восстанавливает исходный текст путём вычитания значений гаммы из зашифрованных данных.

Код также включает механизм проверки пароля, который обеспечивает контроль доступа к процессу шифрования и дешифрования. Если пароль

введён верно, программа генерирует гамму, шифрует пароль и текст, а затем успешно выполняет их дешифрование, демонстрируя корректность работы алгоритма. В случае неверного пароля операция отменяется, что добавляет дополнительный уровень безопасности.

Результаты работы программы подтверждают эффективность аддитивного метода гаммирования для шифрования текстовых данных.

Данная работа может использоваться в образовательных учреждениях как методическое пособие для студентов.

Библиографический список

1. Алексеев М. Е. Шифрование методом гаммирования // 70-я научно-техническая конференция учащихся, студентов и магистрантов, 15-20 апреля 2019 г., Минск : сборник научных работ : в 4 ч. Ч. 4. - Минск : БГТУ, 2019. С. 398-401.
2. Кубашев Д. Ю., Леухин А. Н. Повышение криптостойкости преобразования информации методом гаммирования // Вестник Марийского государственного технического университета. 2008. №3. С. 63.
3. Строгалев С. Ю., Ефремов Е. А. Определение стойкости шифрования методом гаммирования //Безопасные информационные технологии. 2019. С. 338-342.
4. Бурчаловский И. Д., Прохоренко А. А. Криптографический алгоритм "гаммирование" и его исследование //Актуальные вопросы современной техники и технологии. 2015. С. 9-12.
5. Фролов П. Г. и др. Программа реализации процедуры шифрования данных методом гаммирования. Свидетельство о регистрации программы для ЭВМ RU 2021616838, 27.04.2021.