

Гаммирование как метод шифрования: реализация и применение на Python

Екимова Яна Сергеевна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной статье рассматривается реализация метода гаммирования для шифрования и дешифрования текста. Гаммирование — это метод симметричного шифрования, при котором каждый символ исходного текста преобразуется с использованием псевдослучайной последовательности чисел (гаммы). В статье представлены пошаговые реализации функций для генерации гаммы, шифрования и дешифрования текста, а также приведены примеры их использования.

Ключевые слова: Гаммирование, шифрование, дешифрование, python, google colaboratory.

Gamming as an encryption method: implementation and application in Python

Ekimova Yana Sergeevna

Sholom Aleichem Priamurskiy State University

Student

Abstract

This article discusses the implementation of the gamming method for text encryption and decryption. Gamming is a symmetric encryption method in which each character of the source text is converted using a pseudorandom sequence of numbers (gamma). The article presents step-by-step implementations of functions for gamma generation, text encryption and decryption, as well as examples of their use.

Keywords: Gamification, encryption, decryption, python, google coollaboratory.

1 Введение

1.1 Актуальность

В современном мире защита информации является одной из ключевых задач в области информационных технологий. Одним из простых и эффективных методов симметричного шифрования является гаммирование, которое основано на использовании псевдослучайной последовательности чисел (гаммы) для преобразования исходного текста. В данной статье представлена реализация метода гаммирования, включая генерацию гаммы с использованием аддитивного датчика, шифрование текста путем побитового сложения с гаммой и его последующее дешифрование.

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 Google Colaboratory:
<https://colab.research.google.com/drive/1uSbk1IIT4aEnBTfIAZaaHF8kMSfJfnvt?usp=sharing>.

3 Результаты

Реализуем метод гаммирования для шифрования и дешифрования текста. Гаммирование — это метод симметричного шифрования, при котором каждый символ исходного текста складывается (или вычитается) с соответствующим значением гаммы (псевдослучайной последовательности). Ниже приведено подробное описание каждой части кода.

Напишем код, реализующий функцию `generate_gamma`, которая генерирует псевдослучайную последовательность чисел с использованием аддитивного датчика начиная с начального значения `Y0`, каждое следующее значение вычисляется по формуле $(Y_{\text{current}} + Y1) \% m$, где `m` — модуль, определяющий диапазон значений, а `Y1` — 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
```

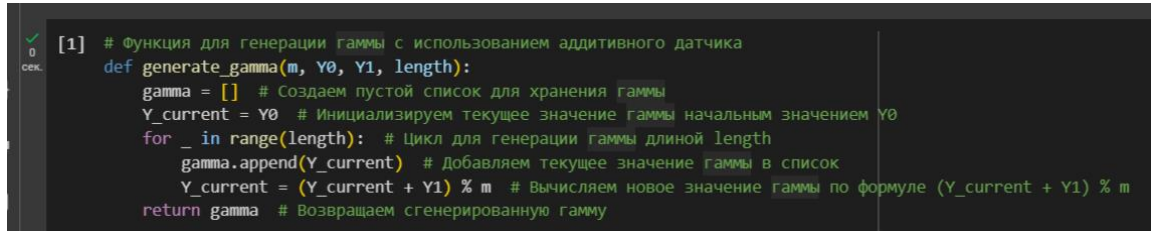
```
    Y_current = Y0 # Инициализируем текущее значение гаммы начальным значением
```

```
    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

```

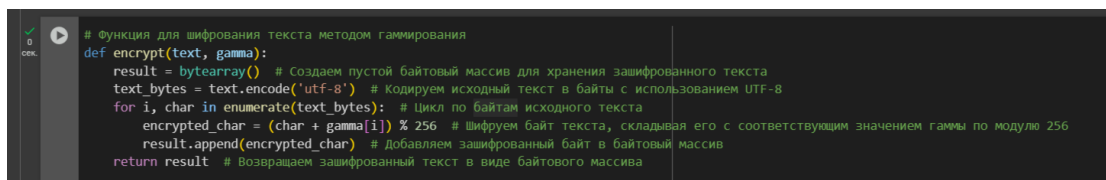
Рисунок 1. Код, реализующий функцию generate_gamma

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
```

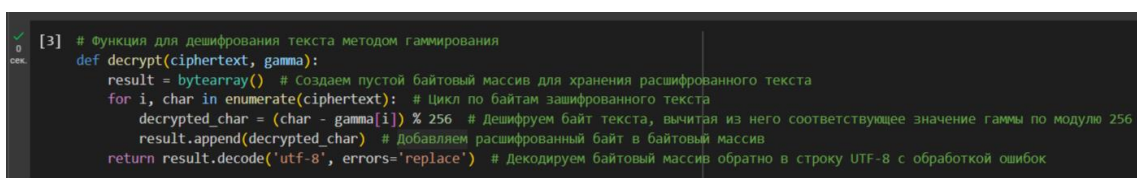
A screenshot of a code editor with a dark background. It shows the Python code for the decrypt function, including comments in Russian. The code is highlighted in green and yellow. On the left side of the editor, there is a small green checkmark icon and the text '[3]'.

Рисунок 3. Код, реализующий функцию decrypt

Затем напомним 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
Y0 = 4003 # Начальное значение
Y1 = 59 # Шаг
```

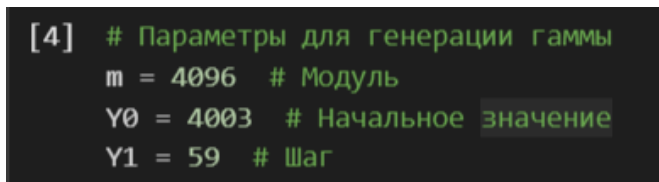
A screenshot of a code editor with a dark background. It shows the Python code for the parameters of the gamma generation, including comments in Russian. The code is highlighted in green and yellow. On the left side of the editor, there is a small green checkmark icon and the text '[4]'.

Рисунок 4. Параметры для генерации гаммы

Создадим переменную plaintext, в которой будет записан текст "Привет, это Яна!", для дальнейшего шифрования.

```
# Пример текста для шифрования  
plaintext = "Привет, это Яна!"
```

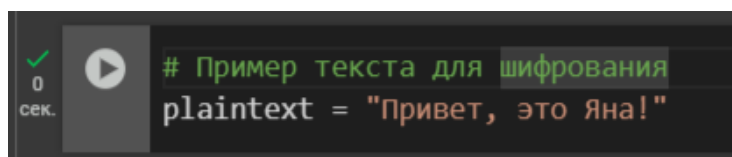


Рисунок 5. Пример текста для шифрования

Далее реализуем код, который будет генерировать гамму (последовательность псевдослучайных чисел) с помощью функции `generate_gamma`. Эта функция принимает следующие параметры: `m` (модуль), `Y0` (начальное значение), `Y1` (шаг) и длину гаммы, которая соответствует длине байтов исходного текста (`plaintext`), закодированного в кодировке UTF-8. Гамма используется, например, в криптографических алгоритмах для шифрования методом XOR или других операций. Результатом выполнения кода будет массив чисел (гамма), длина которого соответствует количеству байтов в исходном тексте. Эта гамма может быть использована для дальнейшего шифрования или преобразования данных.

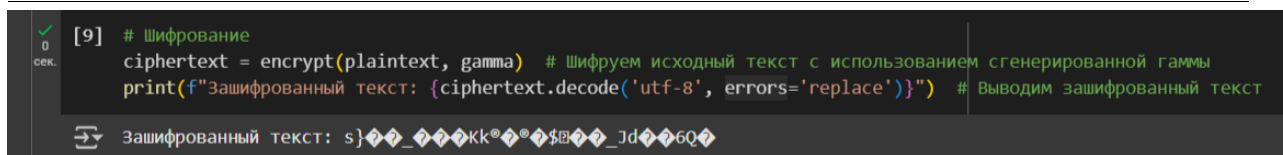
```
# Генерация гаммы  
gamma = generate_gamma(m, Y0, Y1, len(plaintext.encode('utf-8'))) # Генерируем  
гамму длиной, равной длине байтов исходного текста
```



Рисунок 6. Генерация гаммы

Реализуем код, который выполняет шифрование исходного текста (`plaintext`) с помощью функции `encrypt`. Эта функция использует сгенерированную гамму (`gamma`) для преобразования текста в зашифрованное представление. Результатом шифрования является зашифрованный текст (`ciphertext`), который затем декодируется из байтов в строку с использованием кодировки UTF-8. В случае возникновения ошибок при декодировании (например, из-за недопустимых символов), они заменяются на специальный символ замены. Зашифрованный текст выводится на экран с помощью `print`. Результатом выполнения кода будет строка, представляющая зашифрованный текст, который может быть передан или сохранен для дальнейшего использования.

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



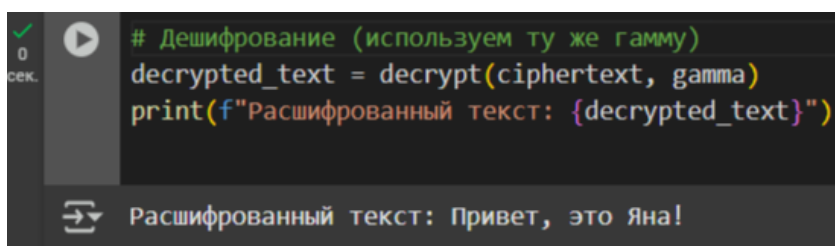
```
[9] # Шифрование
ciphertext = encrypt(plaintext, gamma) # Шифруем исходный текст с использованием сгенерированной гаммы
print(f"Зашифрованный текст: {ciphertext.decode('utf-8', errors='replace')}") # Выводим зашифрованный текст
```

Зашифрованный текст: s}kk\$B_Jd6Q

Рисунок 7. Шифрование

Напишем код, который выполняет дешифрование зашифрованного текста (ciphertext) с помощью функции decrypt. Эта функция использует ту же гамму (gamma), которая была применена при шифровании, чтобы восстановить исходный текст из зашифрованных данных. Результатом дешифрования является восстановление исходного текста (decrypted_text), который затем выводится на экран с помощью print. Если гамма и процесс шифрования/дешифрования были выполнены корректно, результатом будет исходный текст, идентичный тому, что был до шифрования. Этот код демонстрирует процесс обратного преобразования зашифрованных данных в читаемый формат.

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



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

Расшифрованный текст: Привет, это Яна!

Рисунок 8. Дешифрование

4 Выводы

В статье был рассмотрен метод гаммирования для шифрования и дешифрования текста, а также представлены реализации соответствующих функций на языке Python. Генерация гаммы с использованием аддитивного датчика обеспечивает простоту и предсказуемость последовательности, что делает метод удобным для изучения и применения в учебных целях. Шифрование и дешифрование текста выполняются с использованием побитовых операций, что гарантирует сохранение данных в пределах допустимого диапазона значений.

Представленный подход демонстрирует, что метод гаммирования является эффективным инструментом для защиты информации, особенно в случаях, где требуется простота реализации и минимальные вычислительные ресурсы.

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

Библиографический список

1. Алексеев М. Е. Шифрование методом гаммирования. 2019.
2. Кубашев Д. Ю., Леухин А. Н. Повышение криптостойкости преобразования информации методом гаммирования // Вестник Марийского государственного технического университета. 2008. №3. С. 63.
3. Строгалева 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.