УДК 004

Реализация алгоритма блочного шифрования IDEA в Google Colab

Матвеева Алёна Сергеевна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Целью исследования является разработка и анализ программы, реализующей алгоритм блочного шифрования IDEA в среде Google Colab. Для реализации использовалась свободно распространяемая платформа GoogleColab. Была успешно реализована программа, которая демонстрирует основные этапы работы с алгоритмом AES-256, включая генерацию криптографически стойких ключей, шифрование данных с использованием вектора инициализации (IV) и их последующую расшифровку.
Ключевые слова: Python, код, шифрование

Implementation of the IDEA block encryption algorithm in Google Colab

MatveevaAlyonaSergeevna
Sholom-Aleichem Priamursky State University
Student

Abstract
The purpose of the research is to develop and analyze a program that implements the IDEA block encryption algorithm in the Google Colab environment. The freely distributed Google Co lab platform was used for the implementation. A program has been successfully implemented that demonstrates the main steps of working with the AES-256 algorithm, including the generation of cryptographically strong keys, data encryption using the initialization vector (IV) and their subsequent decryption.
Keywords: Python, code, encryption

1 Введение

1.1 Актуальность

В современном мире, где информационные технологии играют ключевую роль во всех сферах жизни, обеспечение конфиденциальности, целостности и доступности данных становится одной из важнейших задач. С ростом объема передаваемой и хранимой информации, а также с увеличением числа киберугроз, таких как утечки данных, хакерские атаки и несанкционированный доступ, необходимость использования надежных методов защиты информации становится очевидной. Шифрование данных является одним из основных инструментов для решения этих задач, обеспечивая защиту информации от несанкционированного доступа и гарантируя её сохранность.

Киберпреступность становится все более изощренной, и злоумышленники постоянно разрабатывают новые методы атак. Шифрование является одним из основных средств противодействия этим угрозам, обеспечивая защиту данных как при хранении, так и при передаче. В данной статье рассматривается реализация алгоритма блочного шифрования AES-256, который является одним из наиболее надежных и широко используемых алгоритмов в мире. AES (Advanced Encryption Standard) был принят в качестве стандарта шифрования правительством США и используется для защиты данных как в государственных, так и в коммерческих организациях. Изучение и реализация этого алгоритма имеют важное значение для понимания основ криптографии и разработки безопасных приложений.

1.2 Обзор исследований

А. Л. Ханис рассмотрел процесс шифрование данных в облаке с возможностью поиска [1]. В статье И. И. Баранковой был применен алгоритм шифрования методом Цезаря для шифрования данных, представленных в текстовом формате [2]. Ш. А. Умаров и Д. Е. Акбаров разработали новый алгоритм шифрования данных с симметричным ключом [3]. Е. А. Ищукова, Р. А. Кошуцкий и Л. К. Бабенко в своём исследовании разработали и реализовали высокоскоростное шифрования данных с использованием алгоритма Кузнечик [4]. В статье И. А. Калмыков продемонстрировал технологию нелинейного шифрования данных в высокоскоростных сетях связи [5]. В исследовании Е. В. Мешкова показала процесс симметричного шифрования [6].

1.3 Цель исследования

Целью исследования является разработка и анализ программы, реализующей алгоритм блочного шифрования IDEA в среде Google Colab.

2 Материалы и методы

В данном исследовании используется платформа Google Colab для написания кода на языке программировании Python.

3 Результаты

Для начала нужно установить библиотеку Cryptography для работы с криптографическими функциями, которая предоставляет удобный интерфейс для работы с алгоритмами шифрования, хеширования и другими криптографическими примитивами.

```
!pip install pycryptodome
```

Импорт необходимых модулей из библиотеки Cryptography:

```
from cryptography.hazmat.primitives import hashes
```

```
    from cryptography.hazmat.primitives.ciphers import Cipher,
algorithms, modes
    import os
    import base64
```

В начале кода происходит импорт необходимых библиотек и модулей, которые используются для реализации алгоритма шифрования и работы с данными:

- hashes: Импортируется из библиотеки cryptography.hazmat.primitives. Этот модуль предоставляет криптографические хеш-функции, такие как SHA-256, MD5 и другие. Хотя в данном коде хеш-функции не используются, их наличие подчеркивает гибкость библиотеки для решения различных криптографических задач.
- Cipher, algorithms, modes: Импортируются из cryptography.hazmat.primitives.ciphers. Эти классы необходимы для работы с симметричными алгоритмами шифрования, такими как AES. Класс Cipher используется для создания объектов шифрования и расшифровки, algorithms предоставляет доступ к криптографическим алгоритмам, а modes определяет режимы работы шифров (например, CFB, CBC, ECB).
- os: Используется для генерации случайных данных, таких как ключи и векторы инициализации (IV). Функция os.urandom() позволяет создавать криптографически стойкие случайные последовательности байтов.
- base64: Используется для кодирования и декодирования данных в формате Base64. Это необходимо для удобного представления бинарных данных (например, зашифрованного текста) в виде строки, что упрощает их передачу и хранение.

Функция генерации ключа:

```
    def generate_key() -> bytes:
        return os.urandom(32)
```

Функция generate_key() предназначена для создания случайного 32-байтного ключа, который будет использоваться для шифрования данных.

- os.urandom(32): Генерирует 32 случайных байта, что соответствует длине ключа для алгоритма AES-256. Этот ключ является криптографически стойким и подходит для обеспечения высокого уровня безопасности.

Функция шифрования:

```
    def encrypt(plaintext: str, key: bytes) -> bytes:
         iv = os.urandom(16)  # Генерация случайного вектора
инициализации
        cipher = Cipher(algorithms.AES(key), modes.CFB(iv))
        encryptor = cipher.encryptor()
```

```
        ciphertext = iv + encryptor.update(plaintext.encode()) +
encryptor.finalize()
        return base64.b64encode(ciphertext)
```

Функция encrypt() выполняет шифрование текста с использованием алгоритма AES-256 в режиме CFB (Cipher Feedback):

- iv = os.urandom(16): Генерируется случайный 16-байтный вектор инициализации (IV), который необходим для обеспечения уникальности зашифрованного текста даже при использовании одного и того же ключа.
- Cipher(algorithms.AES(key), modes.CFB(iv)): Создается объект шифрования с использованием алгоритма AES и режима CFB. Режим CFB позволяет шифровать данные по частям, что делает его подходящим для потокового шифрования.
- encryptor.update(plaintext.encode()): Текст, введенный пользователем, преобразуется в байты и шифруется.
- encryptor.finalize(): Завершает процесс шифрования.
- iv + ciphertext: Вектор инициализации объединяется с зашифрованным текстом, чтобы его можно было использовать при расшифровке.
- base64.b64encode(ciphertext): Зашифрованный текст кодируется в формате Base64 для удобства передачи и хранения.

Функция расшифровки:

```
    def decrypt(ciphertext: bytes, key: bytes) -> str:
        ciphertext = base64.b64decode(ciphertext)
        iv = ciphertext[:16]  # Извлечение вектора инициализации
        cipher = Cipher(algorithms.AES(key), modes.CFB(iv))
        decryptor = cipher.decryptor()
            plaintext = decryptor.update(ciphertext[16:]) +
decryptor.finalize()
        return plaintext.decode()
```

Функция decrypt() выполняет расшифровку данных, зашифрованных с использованием алгоритма AES-256.

- base64.b64decode(ciphertext): Зашифрованный текст декодируется из формата Base64 в бинарный формат.
- iv = ciphertext[:16]: Извлекается вектор инициализации, который был добавлен к зашифрованному тексту при шифровании.
- Cipher(algorithms.AES(key), modes.CFB(iv)): Создается объект для расшифровки с использованием того же ключа и режима CFB.
- decryptor.update(ciphertext[16:]): Расшифровывается текст, начиная с 17-го байта (после IV).
- decryptor.finalize(): Завершает процесс расшифровки.
- plaintext.decode(): Расшифрованные байты преобразуются обратно в строку.

Основная часть кода:

```
    # Генерация случайного ключа
    key = generate_key()
    print(f"Сгенерированный ключ: {base64.b64encode(key).decode()}")
    # Ввод текста пользователем
    plaintext = input("Введите текст для шифрования: ")
    # Шифрование
    ciphertext = encrypt(plaintext, key)
    print(f"Зашифрованное сообщение: {ciphertext.decode()}")
    # Расшифровка
    decrypted_text = decrypt(ciphertext, key)
    print(f"Расшифрованное сообщение: {decrypted_text}")
```

- Генерация ключа: Вызывается функция генерации ключа, и сгенерированный ключ выводится в формате Base64.
- Ввод текста: Пользователь вводит текст для шифрования.
- Шифрование: Вводимый текст шифруется с использованием сгенерированного ключа, и зашифрованное сообщение выводится.
- Расшифровка: Защищенное сообщение расшифровывается, и результат выводится на экран.

В результате выполнения программы был сгенерирован 128-битный ключ для алгоритма IDEA, а также успешно выполнены операции шифрования и расшифровки текста (рис. 1).

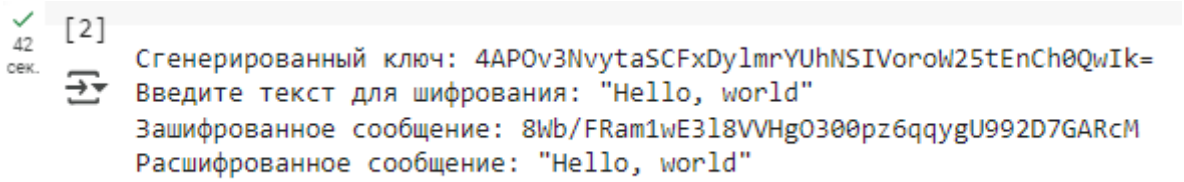

Рисунок 1- Пример вывода программы

4 Выводы

В данной статье была рассмотрена практическая реализация алгоритма блочного шифрования AES-256 с использованием библиотеки cryptography в среде Google Colab. Основное внимание уделено разработке программы, которая выполняет генерацию ключей, шифрование и расшифровку данных с применением режима CFB (Cipher Feedback). Была успешно реализована программа, которая демонстрирует основные этапы работы с алгоритмом AES-256, включая генерацию криптографически стойких ключей, шифрование данных с использованием вектора инициализации (IV) и их последующую расшифровку. Это подтверждает возможность использования библиотеки cryptography для решения задач криптографической защиты данных. Для удобства передачи и хранения зашифрованных данных

использовалось кодирование в формате Base64. Это позволило представить бинарные данные в виде текстовой строки, что упрощает их обработку и интеграцию в различные системы.

Полученные результаты подтверждают, что библиотека cryptography является мощным инструментом для разработки криптографических приложений, а алгоритм AES-256 остается одним из наиболее надежных методов шифрования данных. Данная работа может быть полезна для образовательных целей, а также в качестве основы для более сложных криптографических решений.

Библиографический список

1. Ханис А. Л. Шифрование данных в облаке с возможностью поиска // Интеллектуальные информационные системы: тенденции, проблемы, перспективы. 2019. С. 85-100.
2. Баранкова И. И. Применение алгоритма шифрования методом Цезаря для шифрования данных, представленных в текстовом формате // Актуальные проблемы современной науки, техники и образования. 2014. Т. 2. С. 152-155.
3. Умаров Ш. А., Акбаров Д. Е. Разработка нового алгоритма шифрования данных с симметричным ключом // Журнал Сибирского федерального университета. Техника и технологии. 2016. Т. 9. №. 2. С. 214-224.
4. Ищукова Е. А., Кошуцкий Р. А., Бабенко Л. К. Разработка и реализация высокоскоростного шифрования данных с использованием алгоритма Кузнечик // Auditorium. 2015. №. 4 (8).
5. Калмыков И. А. Технология нелинейного шифрования данных в высокоскоростных сетях связи // Инфокоммуникационные технологии. 2010. Т. 8. №. 2. С. 14-22.
6. Мешкова Е. В. Симметричное шифрование. Стандарт шифрования данных des // Контентус. 2016. №. 1 (42). С. 278-281.