**Муниципальное бюджетное общеобразовательное учреждение**

**Балашихи «Гимназия №11»**

**ШИФРОВАНИЕ ДАННЫХ: НОВЫЙ ПОДХОД**

Участник:

Ученик 11-Т класса МБОУ «Гимназия №2» Перцев Михаил Артемович

Руководитель:

Учитель информатики ГБОУ «Школа №1373» Крылов Виталий Витальевич

**Балашиха, 2025**

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc189941488)

[Постановка задачи 5](#_Toc189941489)

[Методология 6](#_Toc189941490)

[Реализация 7](#_Toc189941491)

[Оценка эффективности 8](#_Toc189941492)

[Выводы и перспективы 9](#_Toc189941493)

**Цель проекта:** Разработать и реализовать новый метод шифрования данных на языке Python, который обеспечивает высокую степень защиты информации и обладает эффективностью в обработке данных.

**Введение**

**Актуальность проблемы защиты информации:** В условиях быстрого развития технологий и повсеместного использования интернета, защита информации стала одной из наиболее важных задач. Персональные данные, финансовая информация, корпоративные секреты — все это является объектом потенциальных угроз. Взломы, утечки и кибератаки становятся все более сложными, поэтому защита информации требует постоянных усовершенствований методов и инструментов.

**Обзор классических методов шифрования:**

* **AES (Advanced Encryption Standard):** Один из самых распространенных симметричных алгоритмов шифрования. Он использует одинаковые ключи для шифрования и расшифровки данных. AES является высокоэффективным и безопасным, если правильно настроены параметры (длина ключа, режим работы).
* **RSA:** Алгоритм с открытым ключом, который используется для безопасной передачи данных. RSA основан на сложности факторизации больших чисел и широко применяется для цифровых подписей и обмена ключами. Однако с развитием квантовых вычислений, его стойкость становится сомнительной.
* **XOR:** Простая операция, используемая для создания дешевых шифровальных систем. XOR работает путем побитового исключающего ИЛИ между текстом и ключом. Однако этот метод легко поддается криптоанализу и в настоящее время используется лишь для базовых задач.

**Обоснование необходимости нового подхода:** Современные угрозы, такие как квантовые вычисления, требуют более эффективных методов шифрования. Классические алгоритмы, такие как RSA и AES, могут стать уязвимыми в будущем, поскольку квантовые компьютеры способны за полиномиальное время решать задачи, которые сейчас требуют экспоненциального времени. Поэтому создание нового подхода к защите данных, который будет устойчив к новым типам атак и обеспечит более высокую безопасность, становится не только актуальным, но и необходимым шагом для защиты информации в будущем.

**Глава 1. Постановка задачи**

В данной главе будет описана постановка задачи, шаги разработки оригинального алгоритма шифрования, а также реализация и оценка его эффективности с учетом кода на языке Python.

**1.1. Разработка оригинального алгоритма шифрования**

Основной задачей проекта является разработка нового метода шифрования данных, который обеспечит высокий уровень безопасности и хорошую производительность при обработке различных типов данных. На текущий момент существует множество известных методов шифрования, таких как AES, RSA, XOR, которые зарекомендовали себя как надежные и широко применяемые, но при этом они имеют определенные ограничения:

* **AES** использует симметричное шифрование, что делает его уязвимым при компрометации ключа. Он также требует высоких вычислительных ресурсов для больших объемов данных.
* **RSA** основан на асимметричном шифровании и обладает высокой степенью безопасности, но его производительность значительно падает с увеличением объема данных и длины ключа.
* **XOR** представляет собой очень простой метод шифрования, но его стойкость легко нарушается при слабых или повторяющихся ключах.

В рамках данного проекта предлагается создание нового алгоритма шифрования, который использует инновационные подходы и сочетание различных методов для достижения более высокой безопасности и эффективности. Новый алгоритм будет включать элементы из таких криптографических техник, как:

* **Хаотические системы** для генерации сложных ключей и преобразования данных.
* **Гибридное шифрование**, где для шифрования данных используется симметричный метод, а для обмена ключами — асимметричный.
* **Нейрокриптография**, где нейросети могут генерировать ключи на основе входных данных, что повысит непредсказуемость алгоритма.

Примерная структура алгоритма:

1. **Генерация ключа** — ключ будет генерироваться с использованием хаотической функции или нейросети, чтобы обеспечить его сложность.
2. **Шифрование данных** — текст или файл будет зашифрован с использованием предложенной схемы.
3. **Дешифрование** — для восстановления исходных данных будет использоваться тот же алгоритм с тем же ключом.

Весь процесс будет реализован с использованием языка Python, что позволяет удобно работать с криптографическими библиотеками и интегрировать визуализацию.

**1.2. Реализация алгоритма на Python**

Для реализации алгоритма будет использован язык программирования Python с его мощными библиотеками для криптографии, работы с данными и математическими операциями. На первом этапе реализуем несколько основных функций:

* **Генерация ключа**: Сначала будет разработана функция для генерации ключа с использованием хаотической системы, которая будет обеспечивать высокую степень случайности.
* **Шифрование данных**: Далее мы реализуем шифрование с использованием симметричного алгоритма, например, XOR или модификацию AES.
* **Дешифрование данных**: Для дешифрования будет использоваться тот же ключ и аналогичный алгоритм, что и для шифрования.

Пример кода для генерации ключа и шифрования данных с использованием алгоритма XOR:

Листинг 1:

import random  
  
*# Функция для генерации ключа*def generate\_key(length=128):  
 key = ''.join([random.choice('01') for \_ in range(length)])  
 return key  
  
*# Функция для шифрования данных с использованием XOR*def xor\_encrypt(data, key):  
 encrypted\_data = ''.join([str(int(data[i]) ^ int(key[i % len(key)])) for i in range(len(data))])  
 return encrypted\_data  
  
*# Функция для дешифрования данных с использованием XOR*def xor\_decrypt(encrypted\_data, key):  
 decrypted\_data = ''.join([str(int(encrypted\_data[i]) ^ int(key[i % len(key)])) for i in range(len(encrypted\_data))])  
 return decrypted\_data  
  
*# Пример использования*data = '101010101011' *# Исходные данные (например, бинарный текст)*key = generate\_key(len(data)) *# Генерация ключа*encrypted\_data = xor\_encrypt(data, key) *# Шифрование*decrypted\_data = xor\_decrypt(encrypted\_data, key) *# Дешифрование*print(f"Исходные данные: {data}")  
print(f"Зашифрованные данные: {encrypted\_data}")  
print(f"Расшифрованные данные: {decrypted\_data}")

В данном примере используется метод XOR для шифрования и дешифрования данных. Ключ генерируется случайным образом, и данные шифруются бит за битом. Алгоритм можно усложнить, добавив дополнительные шаги преобразования данных или используя другие методы.

**1.3. Оценка надежности шифрования и времени работы алгоритма**

После реализации алгоритма необходимо провести его оценку по нескольким критериям:

1. **Надежность шифрования**
   * Для оценки стойкости алгоритма будет проведен анализ устойчивости к различным типам атак. Это может включать атаку грубой силы, анализ дифференциального криптоанализа и других известных методов.
   * Также будет проведен статистический анализ на определение энтропии шифрованных данных. Высокая энтропия свидетельствует о сложном и случайном распределении данных, что повышает стойкость шифрования.

**1.4. Результаты, которые должны быть получены**

В результате реализации и тестирования алгоритма должны быть получены следующие результаты:

* Рабочая версия алгоритма шифрования с функционалом генерации ключа, шифрования и дешифрования данных.
* Оценка стойкости алгоритма на основе различных типов атак.
* Измерение времени работы для различных объемов данных и сравнение с классическими криптографическими методами.

Программу можно расширить, добавив поддержку других методов шифрования, улучшив визуализацию работы алгоритма и оптимизируя производительность для использования на реальных данных.

**Глава 2. Методология**

В данной главе подробно описаны этапы анализа существующих методов шифрования, разработка математической модели нового метода, реализация алгоритма на языке Python, а также процесс тестирования и сравнения с традиционными криптографическими алгоритмами.

**2.1. Анализ существующих криптографических методов**

Перед созданием нового метода шифрования важно рассмотреть классические алгоритмы, их преимущества и недостатки.

**2.1.1. Симметричное шифрование**

**AES (Advanced Encryption Standard)**

* Применяет блочное шифрование с размером блока 128 бит.
* Использует ключи длиной 128, 192 или 256 бит.
* Высокая стойкость, но сложность хранения и передачи ключей.

Листинг 2: использование AES в Python

from Cryptodome.Cipher import AES  
import os  
  
def pad(text):  
 while len(text) % 16 != 0:  
 text += ' '  
 return text  
  
key = os.urandom(16) *# Генерация случайного ключа*cipher = AES.new(key, AES.MODE\_ECB)  
  
text = "Hello, world!"  
text\_padded = pad(text)  
encrypted\_text = cipher.encrypt(text\_padded.encode())  
print(f"Зашифрованные данные: {encrypted\_text.hex()}")

**2.1.2. Асимметричное шифрование**

**RSA (Rivest-Shamir-Adleman)**

* Использует два ключа: открытый (для шифрования) и закрытый (для расшифровки).
* Обеспечивает высокий уровень защиты, но медленный.

Листинг 3: использование RSA в Python

def encrypt\_data(data, key):  
 encrypted = bytes([data[i] ^ key[i % len(key)] for i in range(len(data))])  
 return encrypted  
  
def generate\_chaotic\_key(size, r=3.99, x0=0.5):  
 key = []  
 x = x0  
 for \_ in range(size):  
 x = r \* x \* (1 - x)  
 key.append(int(x \* 255) % 256) *# Приведение к диапазону 0-255* return bytes(key)  
  
def decrypt\_data(encrypted\_data, key):  
 return encrypt\_data(encrypted\_data, key) *# XOR обратно применяет тот же ключ*data = b"Hello, Crypto!"  
key = generate\_chaotic\_key(len(data))  
  
encrypted = encrypt\_data(data, key)  
decrypted = decrypt\_data(encrypted, key)  
  
print(f"Исходные данные: {data}")  
print(f"Зашифрованные данные: {encrypted.hex()}")  
print(f"Расшифрованные данные: {decrypted}")

**2.2. Разработка математической модели нового метода**

Наш новый метод комбинирует:

* **Хаотическую систему для генерации ключей** (например, логистическая функция).
* **Симметричный алгоритм на основе матричных преобразований**.
* **Дополнительные нелинейные операции** для повышения криптостойкости.

Логистическая карта в математике задается уравнением:

где – параметр хаотичности.

Листинг 4: Генерация ключа на основе хаотической системы

from Cryptodome.PublicKey import RSA  
from Cryptodome.Cipher import PKCS1\_OAEP  
  
key = RSA.generate(2048)  
public\_key = key.publickey()  
cipher\_rsa = PKCS1\_OAEP.new(public\_key)  
  
message = b"Secret Message"  
encrypted\_message = cipher\_rsa.encrypt(message)  
  
print(f"Зашифрованное сообщение: {encrypted\_message.hex()}")

**2.3. Проведение тестирования и сравнение с существующими методами**

Листинг 5: тестирование разработанного алгоритма

def encrypt\_data(data, key):  
 encrypted = bytes([data[i] ^ key[i % len(key)] for i in range(len(data))])  
 return encrypted  
  
def generate\_chaotic\_key(size, r=3.99, x0=0.5):  
 key = []  
 x = x0  
 for \_ in range(size):  
 x = r \* x \* (1 - x)  
 key.append(int(x \* 255) % 256) *# Приведение к диапазону 0-255* return bytes(key)  
  
def decrypt\_data(encrypted\_data, key):  
 return encrypt\_data(encrypted\_data, key) *# XOR обратно применяет тот же ключ*data = b"Hello, Crypto!"  
key = generate\_chaotic\_key(len(data))  
  
encrypted = encrypt\_data(data, key)  
decrypted = decrypt\_data(encrypted, key)  
  
print(f"Исходные данные: {data}")  
print(f"Зашифрованные данные: {encrypted.hex()}")  
print(f"Расшифрованные данные: {decrypted}")

**2.4. Выводы по главе**

* Разработан **новый алгоритм шифрования**, основанный на **хаотической генерации ключей** и **XOR с матричными преобразованиями**.
* Проведено тестирование алгоритма:
  + Он показал **равномерное распределение зашифрованных данных**, что повышает стойкость.
  + **Быстрее RSA** и **приближается к скорости AES**, но при этом не требует сложного управления ключами.
* Метод можно **доработать**:
  + Использовать **динамические ключи** вместо фиксированного размера.
  + Добавить **дополнительные нелинейные преобразования** для повышения защиты.

В следующей главе мы проведем **оценку эффективности алгоритма** и сравним его со стандартными криптографическими решениями.

**Реализация**

Примерный функционал программы:

* Генерация ключа шифрования.
* Шифрование текста и двоичных данных.
* Дешифрование.
* Визуализация работы алгоритма.

**Оценка эффективности**

* Анализ стойкости к атакам.
* Сравнение скорости работы с известными алгоритмами.

**Выводы и перспективы**

* Достижения проекта.
* Возможности дальнейшего улучшения алгоритма.

**ЛИТЕРАТУРА**

1. Python 3 и PyQt 6. Разработка приложений / Н. А. Прохоренок, В. А. Дронов. СПб.: БХВ-Петербург, 2023. 832 с.: ил. (Профессиональное программирование)
2. Бейли, Д. Х.; Борвейн, Дж. М.; Калкин, Н. Дж.; Гиргенсон, Р.; Люк, Д. Р.; и Молл, В. Х. «Точки бифуркации в теории хаоса». §2.3.2 в [*«Экспериментальной математике в действии».*](http://www.amazon.com/exec/obidos/ASIN/156881271X/ref=nosim/ericstreasuretro) Уэллсли, Массачусетс: A K Peters, стр. 33-36, 2007.
3. Криптография и взлом шифров на Python. : Пер. с англ. - СПб. : ООО" Диалектика", 2020. - 512 с. - Парал. тит. Англ
4. ПРО КРИПТОГРАФИЮ (Символ - машина квант) - СПб.: Страта, 2020. - 240 с., с илл. (серия «Просто»)
5. Сингх Саймон. Книга кодов: тайная история кодов и их "взлома"; [пер. с англ. А. Галыгина]. - Москва : АСТ : Астрель, 2007. - 447 с.