Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №3
на тему

АСИММЕТРИЧНАЯ КРИПТОГРАФИЯ. КРИПТОСИСТЕМА РАБИНА

Выполнил: студент гр. 253503
Минич С.В.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Криптосистема Рабина это один из классических алгоритмов асимметричной криптографии (шифрования с открытым ключом). В отличие от симметричных методов, в асимметричных системах для шифрования и расшифрования используются разные ключи: открытый ключ (доступный всем) и закрытый ключ (известный только получателю). Криптостойкость системы Рабина основывается на вычислительной сложности задачи факторизации больших чисел, а именно на сложности поиска квадратных корней в кольце остатков по модулю составного числа. Это делает ее безопасность эквивалентной сложности разложения открытого ключа на два больших простых множителя.

Целью данной работы является знакомство с принципами асимметричного блочного шифрования, исследование математических основ Криптосистемы Рабина, а также разработка криптостойкого программного средства на языке программирования, демонстрирующего процессы генерации ключей, шифрования и расшифрования текстовой информации.

1 ХОД ВЫПОЛНЕНИЯ РАБОТЫ

В ходе выполнения работы была реализована программа для криптографической защиты данных на основе асимметричной Криптосистемы Рабина. Основная задача заключалась в разработке программного модуля, способного выполнять генерацию ключей, шифрование и расшифрование текстовых файлов, обеспечивая их конфиденциальность и демонстрируя принцип, согласно которому безопасность системы эквивалентна сложности задачи факторизации.

В программном коде реализованы ключевые математические функции, обеспечивающие работу криптосистемы. Расширенный алгоритм Евклида реализован для нахождения специальных коэффициентов, необходимых для применения Китайской теоремы об остатках на этапе расшифрования. Функция генерации ключей реализована для выбора двух больших случайных простых чисел, каждое из которых при делении на четыре даёт остаток три. Это специальное условие позволяет эффективно вычислять квадратные корни при расшифровании. Функция расшифрования реализует процедуру, которая, используя закрытый ключ и математические теоремы, вычисляет четыре возможных варианта исходного сообщения для каждого зашифрованного блока.

Программный модуль включает логику для создания и сохранения ключей, что является неотъемлемой частью асимметричной криптосистемы. С помощью функции генерации сгенерирована пара ключей: Открытый ключ (произведение двух больших простых чисел) сохранен в файл *public_key.txt*, а Закрытый ключ (сами эти простые числа) в файл *private_key.txt*. Размер блока данных для шифрования динамически определяется на основе длины открытого ключа, что гарантирует выполнение криптографического требования.

Для демонстрации работы криптосистемы с реальными данными реализованы функции для потоковой обработки файлов. В функции шифрования исходный файл считывается блоками байтов, каждый блок конвертируется в большое целое число и шифруется путём возведения его в квадрат по модулю открытого ключа. Полученный шифротекст записывается в выходной файл в виде последовательности чисел. В функции расшифрования шифротекст считывается, и для каждого числа вычисляются четыре возможных варианта исходного сообщения. Для решения проблемы четырех возможных расшифровок используется эвристическая проверка: из четырех вариантов выбирается тот, который при обратном преобразовании в байты успешно считывается как текст. Этот правильный вариант записывается в файл, восстанавливая исходный текст. Результат на рисунке 1.1.

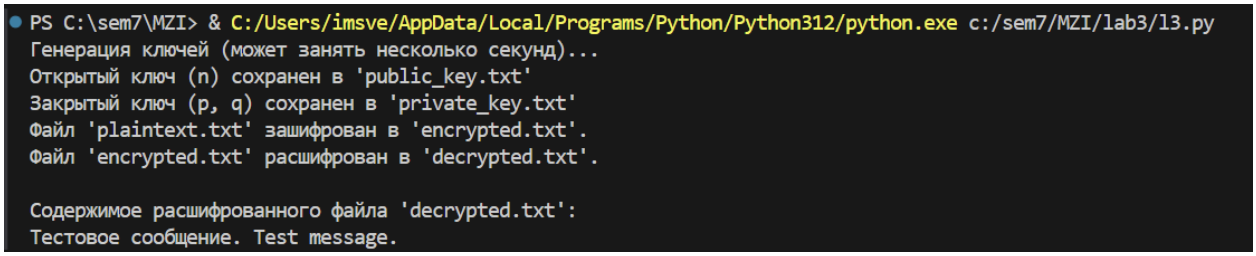

Рисунок 1.1 – Результат выполнения программы

ЗАКЛЮЧЕНИЕ

В ходе работы была подробно изучена и реализована асимметричная криптосистема Рабина. На основе теоретических положений была создана программа на *Python*, включающая генерацию пары ключей, процедуру квадратичного шифрования и многозначное расшифрование с использованием Китайской теоремы об остатках. Реализация позволила продемонстрировать работу ключевых элементов алгоритма: эквивалентность его стойкости сложности задачи факторизации и необходимость применения дополнительного правила для выбора истинного сообщения из четырех корней.

Программа обеспечивает как шифрование, так и последующее успешное восстановление исходной информации, что подтверждает правильность реализации.

Таким образом, цель работы была достигнута: изучение асимметричного алгоритма Рабина и его программное воплощение позволили на практике закрепить понимание теоретических основ современной криптографии с открытым ключом и методов защиты информации.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] КРИПТОГРАФИЧЕСКИЕ АЛГОРИТМЫ ШИФРОВАНИЯ И КОНТРОЛЯ ЦЕЛОСТНОСТИ [Электронный ресурс]. – Режим доступа: belt-spec14.pdf– Дата доступа: 22.09.2025

ЛИСТИНГ ПРОГРАММНОГО КОДА

```
import random
from sympy import isprime

def extended_gcd(a, b):
    """
    Расширенный алгоритм Евклида. Возвращает (gcd, x, y)
    такие, что a*x + b*y = gcd.
    """
    if a == 0:
        return b, 0, 1
    d, x1, y1 = extended_gcd(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return d, x, y

def generate_keys(bit_length=512):
    """
    Генерирует пару ключей для криптосистемы Рабина.
    p и q выбираются такими, что p ≡ 3 (mod 4) и q ≡ 3 (mod 4).
    """
    p = 4
    q = 4
    # Генерируем p
    while (p % 4) != 3 or not isprime(p):
        p = random.getrandbits(bit_length)

    # Генерируем q
    while (q % 4) != 3 or not isprime(q) or q == p:
        q = random.getrandbits(bit_length)

    n = p * q
    return n, (p, q)  # (Открытый ключ), (Закрытый ключ)

def decrypt(c, p, q):
    """
    Расшифровывает шифротекст c с помощью закрытого ключа (p, q).
    Возвращает 4 возможных целочисленных корня.
    """
    n = p * q
    # Вычисляем квадратные корни по модулю p и q
    mp = pow(c, (p + 1) // 4, p)
    mq = pow(c, (q + 1) // 4, q)

    # Используем расширенный алгоритм Евклида для нахождения yp и yq
    _, yp, yq = extended_gcd(p, q)

    # Китайская теорема об остатках для нахождения 4-х корней
    r1 = (yp * p * mq + yq * q * mp) % n
    r2 = n - r1
    r3 = (yp * p * mq - yq * q * mp) % n
    r4 = n - r3

    return [r1, r2, r3, r4]

def encrypt_file(input_filename, output_filename, n):
    """
    Шифрует содержимое входного файла и сохраняет в выходной.
    Работает с файлами любого размера, разбивая их на чанки.
    """
    # Размер чанка в байтах. -1 для гарантии, что число будет < n
    chunk_size = (n.bit_length() - 1) // 8
```

```
        with open(input_filename, 'rb') as f_in, open(output_filename, 'w')
as f_out:
            while True:
                chunk = f_in.read(chunk_size)
                if not chunk:
                    break

                # Конвертируем чанк в число
                m_int = int.from_bytes(chunk, 'big')

                # Шифруем
                c = pow(m_int, 2, n)

                # Записываем шифротекст в файл, каждое число с новой строки
                f_out.write(str(c) + '\n')

        print(f"Файл '{input_filename}' зашифрован в '{output_filename}'.")

    def decrypt_file(input_filename, output_filename, p, q):
        """
        Расшифровывает содержимое входного файла и сохраняет в выходной.
        """
        with open(input_filename, 'r') as f_in, open(output_filename, 'wb')
as f_out:
            for line in f_in:
                if not line.strip():
                    continue
                c = int(line.strip())

                # Получаем 4 возможных корня
                roots = decrypt(c, p, q)

                # Ищем правильный корень.
                 # Эвристика: правильный корень должен декодироваться в UTF-
8.
                # В реальной системе для этого используется паддинг.
                correct_chunk = None
                for r in roots:
                    try:
                        num_bytes = (r.bit_length() + 7) // 8
                        chunk = r.to_bytes(num_bytes, 'big')
                        # Попытка декодирования просто для проверки, что это
"текстовый" чанк
                        chunk.decode('utf-8')
                        correct_chunk = chunk
                        break
                    except (UnicodeDecodeError, AttributeError):
                        # Если не вышло - пробуем следующий корень
                        continue

                if correct_chunk is not None:
                    f_out.write(correct_chunk)
                else:
                      # Если ни один из корней не подошел, возможно это был
последний,
                     # неполный чанк, который не является валидным utf-8 сам
по себе.
                     # Попробуем найти тот, который не вызывает ошибок.
                     if roots:
                         try:
                            # Просто берем первый и пытаемся записать
                            r = roots[0]
                            num_bytes = (r.bit_length() + 7) // 8
```

```
                        f_out.write(r.to_bytes(num_bytes, 'big'))
                    except Exception as e:
                        print(f"Ошибка при записи чанка: {e}")
                        f_out.write(b'[DECRYPTION_ERROR]')

    print(f"Файл '{input_filename}' расшифрован в '{output_filename}'.")

# --- Пример использования ---
if __name__ == '__main__':
    # 1. Генерация ключей
    print("Генерация ключей (может занять несколько секунд)...")
    public_key, private_key = generate_keys(512)
    n = public_key
    p, q = private_key

    print(f"Открытый ключ (n) сохранен в 'public_key.txt'")
    with open("public_key.txt", "w") as f:
        f.write(str(n))

    print(f"Закрытый ключ (p, q) сохранен в 'private_key.txt'")
    with open("private_key.txt", "w") as f:
        f.write(f"{p}\n{q}")

    # 2. Создание файла для шифрования
    with open("plaintext.txt", "w", encoding='utf-8') as f:
        f.write("Тестовое сообщение. Test message.")

    # 3. Шифрование файла
    encrypt_file("plaintext.txt", "encrypted.txt", n)

    # 4. Расшифрование файла
    decrypt_file("encrypted.txt", "decrypted.txt", p, q)

    print("\nСодержимое расшифрованного файла 'decrypted.txt':")
    with open("decrypted.txt", "r", encoding='utf-8') as f:
        print(f.read())
```