Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 7

по дисциплине «Криптографические методы защиты информации»

Криптосистемы с открытым ключом

Студент гр. МКБ 241
_____ В. С. Новиков
«___» _____ 2025 г.

Руководитель
Заведующий кафедрой информационной безопасности киберфизических систем
канд. техн. наук, доцент
_____О.О. Евсютин
«___» _____ 2025 г.

Москва 2025

СОДЕРЖАНИЕ

1. Задание на практическую работу

Целью работы является написать программную реализацию одной из перечисленных ниже асимметричных криптосистем (по выбору студента) с использованием больших чисел. Программная реализация должна быть выполнена студентом самостоятельно без использования готовых библиотечных функций, напрямую реализующих алгоритм шифрования. Варианты криптосистем для реализации:

- RSA;
- Рабина;
- Эль-Гамаля;

В рамках практической работы необходимо выполнить следующее: реализация программы криптосистемы RSA на языке Python

2. Теоретическая часть

2.1 Криптография с открытым ключом

Основной проблемой симметричной криптографии является обеспечение конфиденциальности ключей шифрования при их распределении между пользователями. С данной проблемой сталкиваются все симметричные криптосистемы. Стороны–участники защищенного информационного обмена по открытому каналу связи должны каким-то образом предварительно получить общий секретный ключ, который нельзя передавать в открытом виде. Кроме того, необходимо обеспечить эффективное управление сеансовыми ключами, безопасное хранение долговременных ключей и т. д. Все перечисленные проблемы настолько важны, что в современной криптографии управление ключами выделяют в отдельный раздел.

Для решения проблемы обеспечения конфиденциальности ключей симметричного шифрования в 1976 году была предложена концепция криптографии с открытым ключом.

Криптосистемы с открытым ключом, также называемые асимметричными криптосистемами, используют два разных ключа: открытый ключ зашифрования и закрытый ключ расшифрования. Открытый ключ в общем случае доступен всем желающим, а закрытый ключ известен только законному владельцу. Оба ключа связаны между собой некоторой зависимостью. При этом данная зависимость такова, что, зная один ключ, вычислить другой практически невозможно.

Решение указанной проблемы заключается в том, что отсутствует необходимость в передаче секретной ключевой информации между пользователями. Закрытый ключ должен быть известен лишь его владельцу и не требует передачи.

Базовым понятием в криптографии с открытым ключом является понятие однонаправленной или односторонней функции. Значение такой функции для заданного аргумента достаточно легко вычислить, но практически невозможно вычислить значение аргумента для заданного значения функции. Непосредственно для шифрования однонаправленные функции не используются. Для шифрования применяются однонаправленные функции с лазейкой (ловушкой). Для такой функции вычислить обратную функцию достаточно просто, если известна некоторая секретная информация, и практически невозможно, если эта информация неизвестна.

В асимметричных криптосистемах, построенных на однонаправленных функциях с лазейкой, открытый ключ определяет конкретную реализацию функции, а закрытый ключ дает информацию о лазейке. Основными задачами, приводящими к однонаправленным функциям, являются задачи разложения на множители и дискретного логарифмирования в конечном поле.

2.2 Криптосистемы с открытым ключом RSA

Данная криптосистема является первой криптосистемой с открытым ключом. Она основывается на сложности проблемы факторизации целых чисел, то есть разложения целых чисел на простые множители.

Алгоритм генерации ключей.

1. Пользователь A генерирует два больших простых числа p и q, отличных друг от друга. При этом $|p-q|$ – большое число, хотя p и q имеют приблизительно одинаковый битовый размер.

2. Держа p и q в секрете, Пользователь A вычисляет их произведение $n = pq$, которое называют модулем алгоритма.

3. Пользователь A вычисляет значение функции Эйлера для n по формуле $\varphi(n) = (p-1)(q-1)$.

4. Пользователь A выбирает целое число e, взаимно простое со значением функции $\varphi(n)$. Это число называется экспонентой зашифрования.

5. Пользователь A применяет расширенный алгоритм Евклида к паре чисел e и $\varphi(n)$ и вычисляет значение d, удовлетворяющее соотношению $ed \equiv 1 \bmod \varphi(n)$. Это значение называется экспонентой расшифрования.

6. Пара (e, n) публикуется в качестве открытого ключа пользователя A, d является закрытым ключом и держится в секрете.

Алгоритм зашифрования.

1. Пользователь B получает аутентичную копию открытого ключа пользователя A – пару (e, n).

2. Пользователь B представляет сообщение в виде числа m, меньшего модуля алгоритма. В общем случае сообщение может быть разбито на блоки, каждый из которых представляется своим числом.

3. Пользователь B вычисляет $c = m^e \pmod n$.

4. Зашифрованное сообщение отправляется пользователю A.

Алгоритм расшифрования.

1. Пользователь A получает криптограмму c от пользователя B.

2. Пользователь A вычисляет $m = c^d \pmod n$.

3. Программный код и описание варианта криптосистемы RSA

3.1 Описание

- язык: Python;
- функции генерация ключей, шифрование/расшифрование блока, обработка файла;
- особенности реализация без библиотек;
- generate_keypair - генерирует пару ключей RSA: открытый (e, n) и закрытый (d, n);
- encrypt_block / decrypt_block - шифрование и расшифрование одного блока сообщения с использованием RSA;
- process_file – читает файл и шифрует или расшифровывает его поблочно с использованием RSA, принимает на вход ключ (e, n) для шифрования или (d, n) для расшифрования;
- Интерактивный режим с действиями generate, encrypt, decrypt и справкой (-h).

3.2 Код

Ссылка на код:
https://github.com/vit81g/Cybersecurity_HSE/blob/main/HomeWorks/Cryptographic/practic02/PW01.py

```
import os  # Импортируем модуль os для взаимодействия с операционной системой
import sys  # Импортируем модуль sys для доступа к системным функциям
import random  # Импортируем модуль random для генерации случайных чисел и случайного выбора
import math  # Импортируем модуль math для выполнения математических операций
from typing import Tuple, Optional  # Импортируем аннотации типов Tuple и Optional


def is_prime(n: int, k: int = 5) -> bool:
    """Проверяет, является ли число простым, используя тест Ферма.
    Аргументы: n (int): число для проверки; k (int): количество итераций теста (по умолчанию 5).
    Возвращает: bool: True, если число вероятно простое, False — если составное.
    """
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    # Проводим k итераций теста Ферма
```

```
    for _ in range(k):
        a = random.randint(2, n - 2)
        # Если a^(n-1) ≠ 1 mod n, то n составное
        if pow(a, n - 1, n) != 1:
            return False
    return True

def generate_prime(bits: int) -> int:
    """Генерирует случайное простое число заданной битовой длины.
    Аргументы: bits (int): битовая длина числа.
    Возвращает: int: случайное простое число.
    """
    # Определяем диапазон для чисел заданной длины
    min_val = 1 << (bits - 1)  # 2^(bits-1)
    max_val = (1 << bits) - 1   # 2^bits - 1

    while True:
        n = random.randint(min_val, max_val)
        # Убедимся, что число нечётное
        if n % 2 == 0:
            n += 1
        if is_prime(n):
            return n

def extended_gcd(a: int, b: int) -> Tuple[int, int, int]:
    """Реализует расширенный алгоритм Евклида для нахождения НОД и коэффициентов.
    Аргументы: a (int): первое число; b (int): Второе число.
    Возвращает: Tuple[int, int, int]: НОД(a, b) и коэффициенты x, y, такие что ax + by = НОД.
    """
    if a == 0:
        return b, 0, 1
    gcd, x1, y1 = extended_gcd(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return gcd, x, y

def mod_inverse(e: int, phi: int) -> Optional[int]:
    """Находит мультипликативное обратное e по модулю phi.
    Аргументы: e (int): число, для которого ищется обратное; phi (int): модуль (функция Эйлера).
    Возвращает: Optional[int]: обратное число или None, если оно не существует.
    """
    gcd, x, _ = extended_gcd(e, phi)
    if gcd != 1:
        return None  # e и phi не взаимно простые
    return (x % phi + phi) % phi
```

```python
def generate_keypair(bits: int = 512) -> Tuple[Tuple[int, int], Tuple[int, int]]:
    """Генерирует пару ключей RSA: открытый (e, n) и закрытый (d, n).
    Аргументы: bits (int): битовая длина модуля n (по умолчанию 512).
    Возвращает: Tuple[Tuple[int, int], Tuple[int, int]]: открытый ключ (e, n) и закрытый ключ (d, n).
    Обработка ошибок: ValueError - если не удалось сгенерировать ключи.
    """
    # Генерируем два простых числа p и q
    p = generate_prime(bits // 2)
    q = generate_prime(bits // 2)
    while p == q:
        q = generate_prime(bits // 2)

    # Вычисляем модуль n
    n = p * q

    # Вычисляем функцию Эйлера phi(n)
    phi = (p - 1) * (q - 1)

    # Выбираем открытую экспоненту e
    e = 65537  # Популярное значение, простое и эффективное
    if math.gcd(e, phi) != 1:
        # Если e не взаимно простое с phi, пробуем другое
        for possible_e in range(3, phi, 2):
            if math.gcd(possible_e, phi) == 1:
                e = possible_e
                break
        else:
            raise ValueError("Не удалось найти подходящую открытую экспоненту")

    # Находим закрытую экспоненту d
    d = mod_inverse(e, phi)
    if d is None:
        raise ValueError("Не удалось вычислить закрытую экспоненту")

    return (e, n), (d, n)

def add_padding(data: bytes, block_size: int) -> bytes:
    """Добавляет PKCS#5/PKCS#7 дополнение к данным для кратности блока.
    Аргументы: data (bytes): исходные данные; block_size (int): размер блока в байтах.
    Возвращает: bytes: данные с дополнением.
    """
    padding_length = block_size - (len(data) % block_size)
    padding = bytes([padding_length] * padding_length)
    return data + padding

def remove_padding(data: bytes) -> bytes:
```

```
    """Удаляет PKCS#5/PKCS#7 дополнение из данных.
    Аргументы: data (bytes): данные с дополнением.
    Возвращает: bytes: данные без дополнения.

    Raises:
        ValueError: Если дополнение некорректно.
    """
    if not data:
        return data
    padding_length = data[-1]
    if padding_length > len(data) or padding_length == 0:
        raise ValueError("Некорректное дополнение")
    if not all(b == padding_length for b in data[-padding_length:]):
        raise ValueError("Некорректное дополнение")
    return data[:-padding_length]

def encrypt_block(message: int, e: int, n: int) -> int:
    """Шифрует один блок сообщения с использованием RSA.
    Аргументы:
        message (int): число, представляющее блок сообщения (меньше n).
        e (int): открытая экспонента.
        n (int): модуль.
    Возвращает: int: зашифрованное число.
    Обработка ошибок: ValueError - если message >= n.
    """
    if message >= n:
        raise ValueError("Сообщение должно быть меньше модуля n")
    # c = m^e mod n
    return pow(message, e, n)

def decrypt_block(ciphertext: int, d: int, n: int) -> int:
    """Расшифровывает один блок шифртекста с использованием RSA.
    Аргументы:
        ciphertext (int): число, представляющее блок шифртекста.
        d (int): акрытая экспонента.
        n (int): модуль.
    Возвращает: int: расшифрованное число.
    Обработка ошибок: ValueError - если ciphertext >= n.
    """
    if ciphertext >= n:
        raise ValueError("Шифртекст должен быть меньше модуля n")
    # m = c^d mod n
    return pow(ciphertext, d, n)

def process_file(input_file: str, output_file: str, key: Tuple[int, int], mode: str = "encrypt") -> None:
    """Обрабатывает файл: шифрует или расшифровывает его поблочно с использованием RSA.
```

```
    Аргументы:
        input_file (str): путь к входному файлу.
        output_file (str): путь к выходному файлу.
        key (Tuple[int, int]): ключ (e, n) для шифрования или (d, n) для расшифрования.
        mode (str): режим работы ('encrypt' или 'decrypt').
    Обработка ошибок:
        FileNotFoundError: если входной файл не найден.
        ValueError: если ключ или данные некорректны.
    """
    exp, n = key
    # Определяем размер блока в байтах (чтобы число помещалось в n)
    block_size = (n.bit_length() // 8) - 1  # Минус 1 байт для безопасности

    with open(input_file, 'rb') as f_in:
        data = f_in.read()
        result = b""

        if mode == "encrypt":
            # Добавляем дополнение для шифрования
            data = add_padding(data, block_size)
            # Шифруем поблочно
            for i in range(0, len(data), block_size):
                block = data[i:i + block_size]
                # Преобразуем блок в число
                m = int.from_bytes(block, 'big')
                c = encrypt_block(m, exp, n)
                # Преобразуем шифртекст в байты
                c_bytes = c.to_bytes((n.bit_length() + 7) // 8, 'big')
                result += c_bytes
        else:  # mode == "decrypt"
            # Читаем поблочно, размер блока равен длине n в байтах
            cipher_block_size = (n.bit_length() + 7) // 8
            if len(data) % cipher_block_size != 0:
                raise ValueError("Размер входного файла для расшифровки должен быть кратен размеру
блока")
            temp_result = b""
            for i in range(0, len(data), cipher_block_size):
                block = data[i:i + cipher_block_size]
                # Преобразуем блок в число
                c = int.from_bytes(block, 'big')
                m = decrypt_block(c, exp, n)
                # Преобразуем расшифрованное число в байты
                m_bytes = m.to_bytes(block_size, 'big')
                temp_result += m_bytes
            # Удаляем дополнение после расшифровки
            result = remove_padding(temp_result)
```

```
        with open(output_file, 'wb') as f_out:
            f_out.write(result)

def print_help() -> None:
    """Выводит инструкцию по использованию программы."""
    print("Программа шифрования/расшифрования с использованием криптосистемы RSA")
    print("-----------------------------------------------------------")
    print("Эта программа реализует асимметричную криптосистему RSA.")
    print("Шифрование и расшифрование выполняются с использованием открытого (e, n) и
закрытого (d, n) ключей.")
    print("\nИспользование: python3 rsa_cipher.py [параметры]")
    print("Параметры:")
    print("  -h              Показать эту справку и выйти")
    print("\nИнтерактивный режим:")
    print("1. Запустите программу без параметров: python3 rsa_cipher.py")
    print("2. Выберите действие:")
    print("   - 'generate' для генерации новой ключевой пары.")
    print("   - 'encrypt' для шифрования файла.")
    print("   - 'decrypt' для расшифрования файла.")
    print("3. Следуйте инструкциям на экране:")
    print("   - Для генерации: укажите битовую длину ключей (например, 512).")
    print("   - Для шифрования/расшифрования:")
    print("     - Укажите путь к входному файлу.")
    print("     - Укажите путь к выходному файлу.")
    print("     - Введите ключ (e и n для шифрования, d и n для расшифрования).")
    print("\nПример ключей:")
    print("  Открытый ключ: e=65537, n=12345678901234567890")
    print("  Закрытый ключ: d=98765432109876543210, n=12345678901234567890")
    print("\nПример использования:")
    print("1. Генерация ключей:")
    print("   - Действие: generate")
    print("   - Битовая длина: 512")
    print("2. Шифрование файла plaintext.txt в encrypted.bin:")
    print("   - Действие: encrypt")
    print("   - Входной файл: plaintext.txt")
    print("   - Выходной файл: encrypted.bin")
    print("   - Ключ: e=65537, n=12345678901234567890")
    print("3. Расшифрование файла encrypted.bin в decrypted.txt:")
    print("   - Действие: decrypt")
    print("   - Входной файл: encrypted.bin")
    print("   - Выходной файл: decrypted.txt")
    print("   - Ключ: d=98765432109876543210, n=12345678901234567890")
    print("\nПримечания:")
    print("- Входной файл для шифрования может быть текстовым (UTF-8) или бинарным.")
    print("- Для расшифровки используйте тот же модуль n, что при шифровании.")
```

```
    print("- Ключи вводятся как два числа через пробел (например, '65537
12345678901234567890').")
    print("- Результат шифрования — бинарные данные, не открывайте их как текст.")

def main() -> None:
    """Основная функция программы: обрабатывает параметры командной строки и запускает
обработку."""
    if len(sys.argv) > 1 and sys.argv[1] == "-h":
        print_help()
        sys.exit(0)

    print("Программа шифрования/расшифрования с использованием криптосистемы RSA")
    print("-----------------------------------------------------------")

    while True:
        action = input("Выберите действие (generate/encrypt/decrypt): ").strip().lower()
        if action in ["generate", "encrypt", "decrypt"]:
            break
        print("Ошибка: введите 'generate', 'encrypt' или 'decrypt'")

    if action == "generate":
        while True:
            try:
                bits = int(input("Введите битовую длину ключей (например, 512): ").strip())
                if bits < 64:
                    print("Ошибка: битовая длина должна быть не менее 64")
                    continue
                break
            except ValueError:
                print("Ошибка: введите целое число")
        try:
            public_key, private_key = generate_keypair(bits)
            print(f"Открытый ключ: e={public_key[0]}, n={public_key[1]}")
            print(f"Закрытый ключ: d={private_key[0]}, n={private_key[1]}")
        except Exception as e:
            print(f"Ошибка при генерации ключей: {e}")
        return

    while True:
        input_file = input("Введите путь к входному файлу: ").strip()
        if os.path.exists(input_file):
            break
        print("Ошибка: файл не найден")

    output_file = input("Введите путь к выходному файлу: ").strip()
```

```
    while True:
        key_input = input(f"Введите ключ ({'e n' if action == 'encrypt' else 'd n'}): ").strip()
        try:
            exp, n = map(int, key_input.split())
            if exp <= 0 or n <= 0:
                raise ValueError("Ключи должны быть положительными числами")
            break
        except ValueError:
            print("Ошибка: введите два числа через пробел (например, '65537
12345678901234567890')")

    try:
        process_file(input_file, output_file, (exp, n), action)
        print(f"Операция завершена! Результат сохранён в {output_file}")
        if action == "decrypt":
            with open(output_file, 'rb') as f:
                decrypted_data = f.read()
                try:
                    decrypted_text = decrypted_data.decode('utf-8')
                    print(f"Расшифрованный текст: {decrypted_text}")
                except UnicodeDecodeError:
                    print("Результат не является текстом в кодировке UTF-8")
    except Exception as e:
        print(f"Произошла ошибка: {e}")

# запуск
if __name__ == "__main__":
    main()
```

3.3 Запуск

Сохранить код в файл, например <script>.py.

Запустить в терминале: python <script>.py.

Дополнительные инструкции:

- generate - генерация ключей, если пользователь не может сам предоставить ключи
- выбрать encrypt или decrypt;
- указать путь к файлу (например, plaintext.txt);
- указать имя выходного файла (например, encrypted.bin);
- предоставить ключ, например (2048 битный):

Открытый ключ: e=65537,
n=20550052083448917747637528851060681894491668833317099787075539249786242111300001572214832735775672252544567845197630828253961601227397210317954130166112013988531220202513714738620183649029238006670786314084427362301508078235214439206613697012423886086397750138904800570101364651768702438201340296418202278329727967514465481009564791443238873107136733212734411763915813542424957443510514489562843268025062159580225469258442193767639683550807412564748207328022079498432835197427361463497704340552190606945901459773537174283090128721384765490030497091167175743769032367060168825739244178512672899730836826626830414306879

Закрытый ключ:
d=18829839291746508989782584571082670986559030854471903799589136022230398477587872414255655898440039892542439288828645328096779652948814103129582731443386065154420988210951845274713225327797347628646221049896759807949217413157496716187746755562400884742577037300934924989473992534039738622708221098619245083173798306952680590532345017234378512285807728119360008940602400755877540611254129977433030762018451849071720853291044288479133117789370431528048275085741283552625710985059227862819799275465547912902934137101550941657275486987920119674259639215773884221769583297762935588791125270521374908409851312752322695655553,
n=20550052083448917747637528851060681894491668833317099787075539249786242111300001572214832735775672252544567845197630828253961601227397210317954130166112013988531220202513714738620183649029238006670786314084427362301508078235214439

9206613697012423886086397750138904800570101364651768702438201340296418202278329727967514465481009564791443238873107136733212734411763915813542424957443510514489562843268025062159580225469258442193767639683550807412564748207328022079498432835197427361463497704340552190606945901459773537174283090128721384765490030497091167175743769032367060168825739244178512672899730836826626830414306879

3.4 Пример работы

Пример вывода результата работы программы «Криптосистема RSA»:

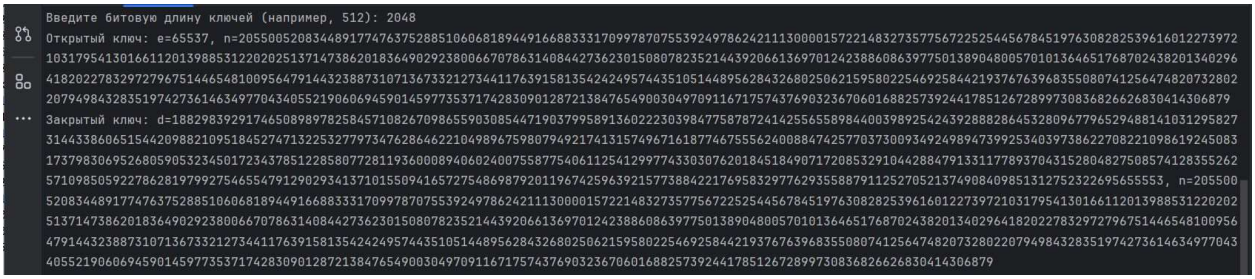

Рисунок 1 – Пример работы программы «Криптосистема RSA» - generate

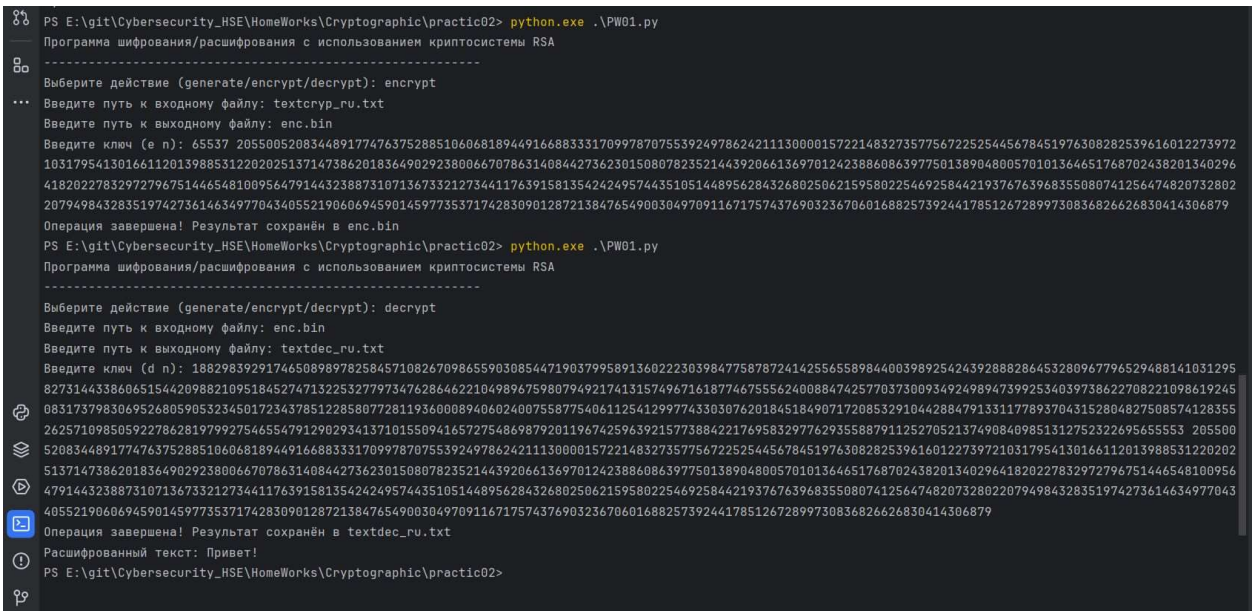

Рисунок 2 – Пример работы программы «Криптосистема RSA» - encrypt и decrypt

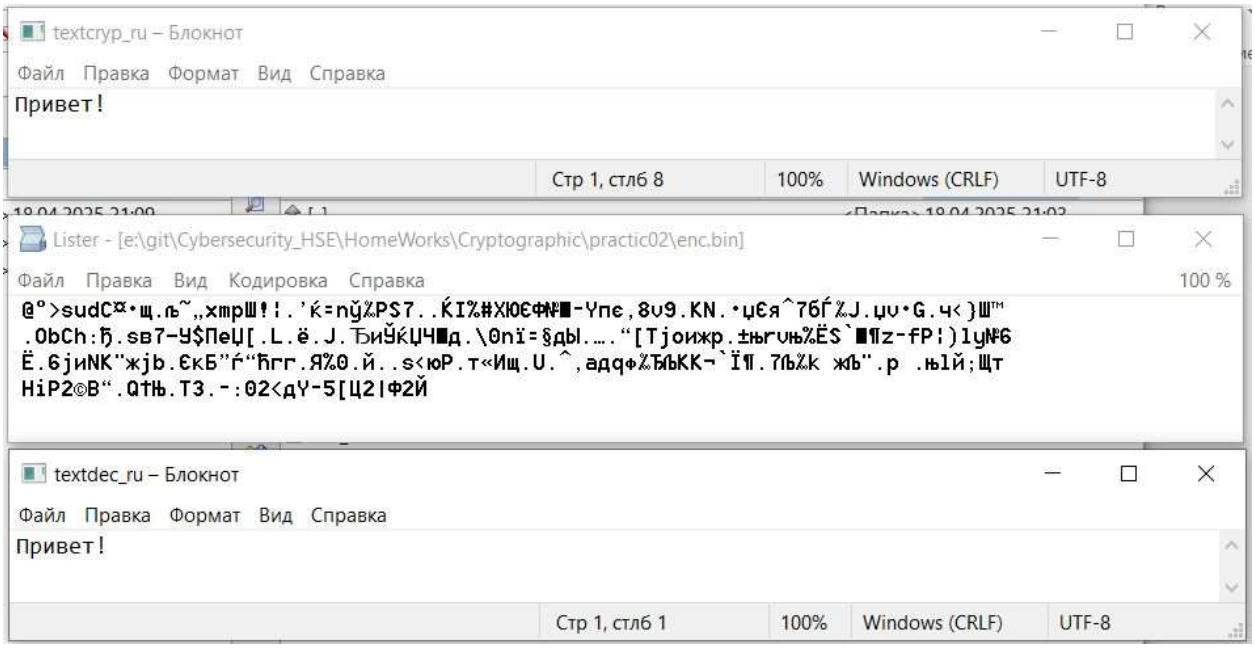

Рисунок 3 – Пример файлов: textcryp_ru (исходное сообщение), enc.bin (зашифрованное сообщение, textdec_ru (расшифрованное сообщение)

4. Реализованная атака на криптосистему

4.1 Описание атаки

Атака основана на факторизации малого модуля (n). Если (n) можно разложить на (p) и (q), то:

1. Вычисляется (\phi(n) = (p - 1)(q - 1)).
2. Находится (d), обратное (e) по модулю (\phi(n)).
3. Шифртекст расшифровывается: (m = c^d \mod n).

Для малых (n) факторизация выполняется методом пробного деления. Для больших (n) (например, 2048 бит) атака неосуществима.

4.2 Код атаки

Ссылка на код:

https://github.com/vit81g/Cybersecurity_HSE/blob/main/HomeWorks/Cryptographic/practic02/attack_in_RSA.py

Реализован в rsa_attack.py:

- factorize_small_n: Факторизует (n) на (p, q).
- attack_rsa: Выполняет атаку, вычисляя (\phi(n)), (d) и расшифровывая (c).
- Использует extended_gcd и mod_inverse из основного скрипта.

4.3 Числовые результаты

Параметры**:**

- Открытый ключ: (e = 7, n = 77) ((p = 7, q = 11)).
- Шифртекст: (c = 33) (соответствует (m = 33), символ !).

Шаги атаки**:**

1. Факторизация: (n = 77 = 7 \cdot 11).
2. (\phi(n) = (7 - 1)(11 - 1) = 60).
3. (d = 43) (так как (7 \cdot 43 \equiv 1 \pmod{60})).
4. Расшифрование: (m = 33^{43} \mod 77 = 33).
5. (m = 33) декодируется как !.

4.4 Вывод программы:

Открытый ключ: e = 7, n = 77

Шифртекст: c = 33

Факторизация: n = 77 = 7 * 11

phi(n) = 60

Закрытая экспонента: d = 43

Расшифрованное сообщение (число): 33

Расшифрованный текст: !

Атака успешна для малых (n), но не работает для больших чисел.

5. ПРИЛОЖЕНИЕ А.

Пример списка использованных источников

1. Стинсон Д. Криптография. Теория и практика. – М.: Техносфера, 2006. – 608 с.
2. Столлингс У. Криптография и сетевая безопасность. – М.: Вильямс, 2014. – 944 с.
3. Python Software Foundation. Python Documentation. – URL: https://docs.python.org/3/
4. Сейтц Д. Black Hat Python. – СПб.: Питер, 2020. – 224 с.
5. Мартин Р. Чистый код: создание, анализ и рефакторинг // «Библиотека программиста (Питер)», 2024г.
6. Шоу Зед. Легкий способ выучить Python 3 // «Бомбора», 2021г.