Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий
Дисциплина «Средства и методы защиты информации в интеллектуальных системах»

ОТЧЁТ
к лабораторной работе №5
на тему
«АСИММЕТРИЧНОЕ ШИФРОВАНИЕ И ЭЛЕКТРОННАЯ ЦИФРОВАЯ ПОДПИСЬ»

БГУИР 6-05-0611-03 130

Выполнил студент группы 321701
СЕМЕНЯКО Владимир Дмитриевич

_____
(дата, подпись студента)

Проверил
САЛЬНИКОВ Даниил Андреевич

_____
(дата, подпись преподавателя)

Минск 2025

1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

а) Разработать программное обеспечение на языке Python, реализующее алгоритм RSA, включая:

– Генерацию пары ключей (открытого и секретного) на основе двух простых чисел p и q длиной не менее 1024 бит.

– Функцию шифрования сообщения с использованием открытого ключа получателя.

– Функцию расшифрования сообщения с использованием секретного ключа получателя.

– Функцию создания цифровой подписи с использованием секретного ключа отправителя.

– Функцию проверки цифровой подписи с использованием открытого ключа отправителя.

б) Для повышения производительности использовать алгоритм быстрого возведения в степень (метод последовательного возведения в квадрат и умножения).

в) Обеспечить обмен данными через файлы:

– $public\_key.txt$ — для хранения открытого ключа (e, n).

– $private\_key.txt$ — для хранения секретного ключа (d, n).

– $message.txt$ — для хранения исходного сообщения.

– $encrypted\_message.txt$ — для хранения зашифрованного сообщения.

г) Выполнить тестирование на 10 наборах тестовых данных.

д) Описать шаги, выполняемые при шифровании, расшифровании, создании и проверке подписи.

е) Сделать выводы, содержащие оценку стойкости алгоритма, возможные угрозы и предложения по защите.

2 ВЫПОЛНЕНИЕ РАБОТЫ

Листинг 1 – Код программы

```
import random
from Crypto.Util.number import getPrime, bytes_to_long, long_to_bytes


def mod_exp(base, exp, mod):
    result = 1
    base = base % mod
    while exp > 0:
```

```
        if exp % 2 == 1:
            result = (result * base) % mod
        exp = exp >> 1
        base = (base * base) % mod
    return result


def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    else:
        gcd, x1, y1 = extended_gcd(b % a, a)
        x = y1 - (b // a) * x1
        y = x1
        return gcd, x, y


def mod_inverse(a, m):
    gcd, x, y = extended_gcd(a, m)
    if gcd != 1:
        raise ValueErrorОбратный(" элемент не существует")
    else:
        return x % m


def generate_prime(bits):
    return getPrime(bits)


def generate_keys(bits=1024):

    print(fГенерация" простых чисел длиной {bits} бит...")
    p = generate_prime(bits)
    q = generate_prime(bits)
    printПростые(" числа p и q успешно сгенерированы.")

    n = p * q
    phi_n = (p - 1) * (q - 1)
    e = 65537
    if extended_gcd(e, phi_n)[0] != 1:
        e = 65539
        if extended_gcd(e, phi_n)[0] != 1:
            raise ValueErrorНе(" удалось найти подходящую экспоненту e.")

    d = mod_inverse(e, phi_n)
    with open('public_key.txt', 'w') as f:
        f.write(f"{e}\n{n}")
    printОткрытый(" ключ (e, n) сохранен в 'public_key.txt'.")
    with open('private_key.txt', 'w') as f:
        f.write(f"{d}\n{n}")
    printСекретный(" ключ (d, n) сохранен в 'private_key.txt'.")

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

```
def encrypt(message, e, n):
    message_bytes = message.encode('utf-8')
    message_int = bytes_to_long(message_bytes)

    if message_int >= n:
        raise ValueError(
            Сообщение" слишком длинное для данного ключа. Увеличьте длину
    ключа или разбейте сообщение на части.")
    ciphertext = mod_exp(message_int, e, n)
    return ciphertext


def decrypt(ciphertext, d, n):
    message_int = mod_exp(ciphertext, d, n)
    try:
        message_bytes = long_to_bytes(message_int)
        message = message_bytes.decode('utf-8')
    except:
        raise ValueErrorОшибка(" при расшифровке. Возможно, используется
    неверный ключ.")

    return message


def sign(message, d, n):
    message_bytes = message.encode('utf-8')
    message_int = bytes_to_long(message_bytes)
    signature = mod_exp(message_int, d, n)
    return signature


def verify(message, signature, e, n):
    message_bytes = message.encode('utf-8')
    message_int = bytes_to_long(message_bytes)
    recovered_message_int = mod_exp(signature, e, n)

    return message_int == recovered_message_int


def load_public_key(filename='public_key.txt'):
    with open(filename, 'r') as f:
        e = int(f.readline().strip())
        n = int(f.readline().strip())
    return e, n


def load_private_key(filename='private_key.txt'):
    with open(filename, 'r') as f:
        d = int(f.readline().strip())
        n = int(f.readline().strip())
    return d, n
```

Результат программы:

```
--- Шаг 1: Генерация ключей ---
Генерация простых чисел длиной 1024 бит...
Простые числа p и q успешно сгенерированы.
Открытый ключ (e, n) сохранен в 'public_key.txt'.
Секретный ключ (d, n) сохранен в 'private_key.txt'.
Открытый ключ (e, n): (65537, ...4090868111...)
Секретный ключ (d, n): (...7094938137..., ...4090868111...)

--- Шаг 2: Шифрование и расшифрование ---
Исходное сообщение записано в 'message.txt': "Секретное сообщение для Боба: Встреча в 18:00."
Зашифрованное сообщение (целое число): 628189704193182256627367642052253358287769637989445733510995063332770724746694602905224907251630498577711447708844930079649545544908235883369245379719600728714115056146780096674529149656949921391060948771843810403231730262412579052633251675297432663855881700228482782236836720845023958710980454560144670416296938353344552627186251037131134952615679946488554556837140951466334902650612088135494448330071699675440385446120002193683208091075773267733803597938262167708557080003871238349842843573222682985012090025492772630357544067832233685506391736431104965696120450061094467227825086821656900819848067029006756840400
Зашифрованное сообщение сохранено в 'encrypted_message.txt'.

Расшифрованное сообщение: "Секретное сообщение для Боба: Встреча в 18:00."
Успешно? True

--- Шаг 3: Создание и проверка цифровой подписи ---
Цифровая подпись (s): 108040865197406872560653761917091369848246622309434747957231796683057018956394860863513296243956381289215276780650466471250303034241862255986236740374509958933042653040254844272107836794392042341237285814204371505706202358034403748018497006563726551116284868420349882291323037168557266296447776205006372921362051498359715330648786596801203572692105955208458678351903138680910071047846079618568919381385231425169703576845826977695207633685418981412422560458828264098829873248736759702302711014165297798605649179065720672610683250070456701623822486102746436604837180075742391056369837741459317087228561387417873843211 69
Подпись верна: True
Подпись для поддельного сообщения верна: False

--- Шаг 4: Тестирование на 10 наборах данных ---
ВСЕ 10 ТЕСТОВ УСПЕШНО ПРОЙДЕНЫ!
```

Рисунок 1 – Результат выполнения программы

Участники: Алиса (отправитель) и Боб (получатель).

а) Шаг 1: Генерация ключей. Алиса генерирует свою пару ключей RSA.

– Она выбирает два больших простых числа p и q (длиной 1024 бит каждое).

– Вычисляет $n = p \cdot q$ и $\phi(n) = (p-1) \cdot (q-1)$.

– Выбирает открытую экспоненту $e = 65537$.

– Вычисляет секретную экспоненту d, такую что $d \cdot e \equiv 1 \pmod{\phi(n)}$.

– Сохраняет открытый ключ (e, n) в файл `public_key.txt` и секретный ключ (d, n) в файл `private_key.txt`.

б) Шаг 2: Шифрование сообщения. Алиса хочет отправить Бобу зашифрованное сообщение ”Секретное сообщение для Боба: Встреча в 18:00.”.

– Она читает открытый ключ Боба (e, n).

– Преобразует текст сообщения в большое целое число m.

– Вычисляет криптограмму $c = m^e \mod n$.

– Сохраняет c в файл `encrypted_message.txt`.

в) Шаг 3: Расшифрование сообщения. Боб получает файл `encrypted_message`

– Он читает свой секретный ключ (d, n).

– Вычисляет исходное сообщение $m = c^d \mod n$.

– Преобразует число m обратно в текстовую строку.

г) Шаг 4: Создание цифровой подписи. Алиса хочет подписать свое сообщение.

– Она читает свой секретный ключ (d, n).

– Преобразует текст сообщения в число m.
– Вычисляет подпись $s = m^d \mod n$.
– Отправляет Бобу пару (сообщение, s).

д) Шаг 5: Проверка цифровой подписи. Боб хочет проверить подлинность сообщения.

– Он получает сообщение и подпись s.
– Он получает открытый ключ Алисы (e, n).
– Вычисляет $m' = s^e \mod n$.
– Сравнивает m' с числом, полученным из текстового сообщения.
– Если $m' = m$, подпись верна.

ВЫВОД

Оценка стойкости алгоритма

Стойкость алгоритма RSA основана на вычислительной сложности задачи факторизации больших целых чисел. Для взлома системы злоумышленнику необходимо разложить модуль n на множители p и q. На сегодняшний день для обеспечения надежной защиты рекомендуется использовать ключи длиной не менее 2048 бит. В данной работе для демонстрации использовались ключи длиной 1024 бит, что является минимальным порогом.

Возможные угрозы

а) **Атака на основе факторизации.** Прямая атака на разложение числа n.

б) **Атака "человек посередине"(MitM).** Злоумышленник может перехватывать и подменять открытые ключи.

в) **Атака на генератор случайных чисел.** Если p и q генерируются с использованием слабого ГСЧ.

г) **Атака по сторонним каналам.** Анализ времени выполнения или энергопотребления для вычисления d.

Предложения по защите

а) **Использование ключей достаточной длины.** 3072 или 4096 бит для долгосрочной безопасности.

б) **Аутентификация ключей.** Использование PKI и цифровых сертификатов для защиты от MitM.

в) **Безопасная генерация ключей.** Использование криптографически стойких генераторов случайных чисел.

г) **Гибридные системы.** Использование RSA для обмена сеансовым ключом симметричного шифра (например, AES).