Министерство науки И ВЫСШЕГО ОБРАЗОВАНИЯ   
Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Национальный исследовательский Томский политехнический Университет»



Инженерная школа автоматизации и робототехники

Отделение автоматизации и робототехники

15.04.04 «Автоматизация технологических процессов и производств»

Вариант 1

Лабораторная работа №5

**«Алгоритм RSAЭлектронная цифровая подпись»**

по дисциплине:

**«Информационная безопасность автоматизированных систем»**

**Исполнитель:**

студент группы 8ТМ22 Гао Аозе \_\_\_\_\_\_\_\_\_\_

**Руководитель:**

К.т.н., доцент ОАР Суходоев Михаил Сергеевич \_\_\_\_\_\_\_\_\_\_

Томск – 2023

## Цель работы

Ознакомиться с концепцией электронной цифровой подписи, программно реализовать алгоритм формирования электронной цифровой подписи и проверки ее подлинности.

**Электронная подпись на основе RSA**

Для осуществления подписи сообщения M необходимо вычислить хэш-функцию m = h(M), которая ставит в соответствие сообщению M число m. На следующем шаге достаточно снабдить подписью только число m, и эта подпись будет относиться ко всему сообщению M.

Протокол на базе алгоритма RSA состоит из следующих этапов.

Этап 1. Выработка ключей (e,n) и (d,n) по алгоритму RSA (выполняет отправитель А) (см. лабораторную работу 4).

Этап 2. Отправка сообщения и электронной подписи (выполняет отправитель А).

1. Вычисление хэш-образа m = h(M), где M – исходное сообщение, m(M) – хеш-функция (для MD5 длина хэш-образа 128 бит).
2. Выработка цифровой подписи s = m d modn, где d – закрытый ключ отправителя A, n – часть открытого ключа отправителя A.

3)Отправка получателю B исходного сообщения M и цифровой подписи s <M, s>.

Этап 3. Получение сообщения и проверка электронной подписи (выполняет получатель B).

1. Вычисление хэш-образа по полученному сообщению m' = h(M), где M – полученное сообщение.
2. Вычисление хэш-образа из цифровой подписи m = s^e modn, где e и n – открытый ключ отправителя A.2)

3)Если m' = m, то получатель B делает вывод, что полученное сообщение M действительно отправлено А и не изменялось.

## Задачи

1. Используя программу, реализующую алгоритм RSA (см. лаб. раб. 4), разработать программу для вычисления ЭЦП.

При этом:

1.1 Значения (e,n) и (d,n) должны быть получены в соответствии с указаниями к лабораторной работе 4.

1.2 Значение хэш-функции m = h(M) принимать равным значению, указанному в таблице 1 для Вашего варианта.

2. Осуществить проверку правильности работы программы. Для этого необходимо вручную рассчитать ЭЦП для некоторых заданных параметров и сравнить полученный результат с результатом работы программы.

3. Реализовать в программе функцию проверки ЭЦП. Для заданных в таблице 1 открытых ключей пользователя проверить подлинность подписанных по алгоритму RSA хэш-значений некоторых сообщений. Результаты проверки привести в отчете.

4. Оформить отчет по лабораторной работе.

**Таблица 1. Варианты заданий**

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Значение хэш-образа *m* | ЭЦП по алгоритму RSA | |
| Открытые ключи (*e*,*n*) | Проверяемые сообщения <*m*, *s*>, где *m* – хэш-значение сообщения *M* |
| 1 | 15 | *n* = 143, *e* = 37 | <46,85>, <16,74>,<129,116> |

## Ход работы

На основе программы алгоритма RSA разработана программа для вычисления ЭЦП.

Сначала мы добавляем функцию цифровой подписи, проверяем функцию цифровой подписи и добавляем простую хэш-функцию, а также изменяем основную функцию, основываясь на реализации алгоритма RSA на основе четвертого эксперимента. Результаты следующие：

Функция проверки простых чисел is\_prime: Используется для проверки того, является ли число простым.

Функция наибольшего общего делителя gcd: Используется для вычисления наибольшего общего делителя двух чисел.

Функция модулярного инверса mod\_inverse: Используется для вычисления модальной обратной величины, т. е. мультипликативной обратной величины a - x, удовлетворяющей (a \* x) % m = 1, по модулю m.

Функция генерации пары ключей RSA generate\_keypair: Введить два простых числа p и q. Вычислите n(p \* q) и функцию Эйлера φ((p-1) \* (q-1)). Выберить открытый ключ e, равный 37, обычно также 65537. Вычислить закрытый ключ d, такой, что (e \* d) % φ = 1. Выведить открытый и закрытый ключи и верните их.

Функция шифрования encrypt: Вводит сообщение с открытым текстом и открытый ключ и выполняет RSA-шифрование, преобразуя каждый символ сообщения в его ASCII-код. Вернит зашифрованный шифртекст и открытый ключ n.

Расшифровать: Введить шифртекст и закрытый ключ, расшифруйте каждое число в шифртексте методом RSA и преобразуйте результат обратно в символы. Возвращать расшифрованный открытый текст.

функция цифровой подписи sign: Введить сообщение, которое нужно подписать, закрытый ключ и открытый ключ n. Вычислять хэш-значение сообщения с помощью простой хэш-функции. Выполнять RSA-подпись хэша. Возвращать подписанное значение.

Проверка функции цифровой подписи verify: Введить проверяемое сообщение, подпись, открытый ключ и открытый ключ n. Вычислить хэш сообщения, используя ту же хэш-функцию. Выполняет проверку RSA для подписанного значения. Возвращает результат проверки.

Простая хэш-функция hash\_function:

Простая хэш-функция для примера, которая складывает ASCII-код каждого символа в сообщении.

Главная функция main:

Генерирует пару ключей RSA из введенных пользователем простых чисел p и q.

Предоставляет пользователю простое меню для выбора шифрования, расшифровки, подписи, проверки или выхода.

Выполняет соответствующее действие в зависимости от выбора пользователя.

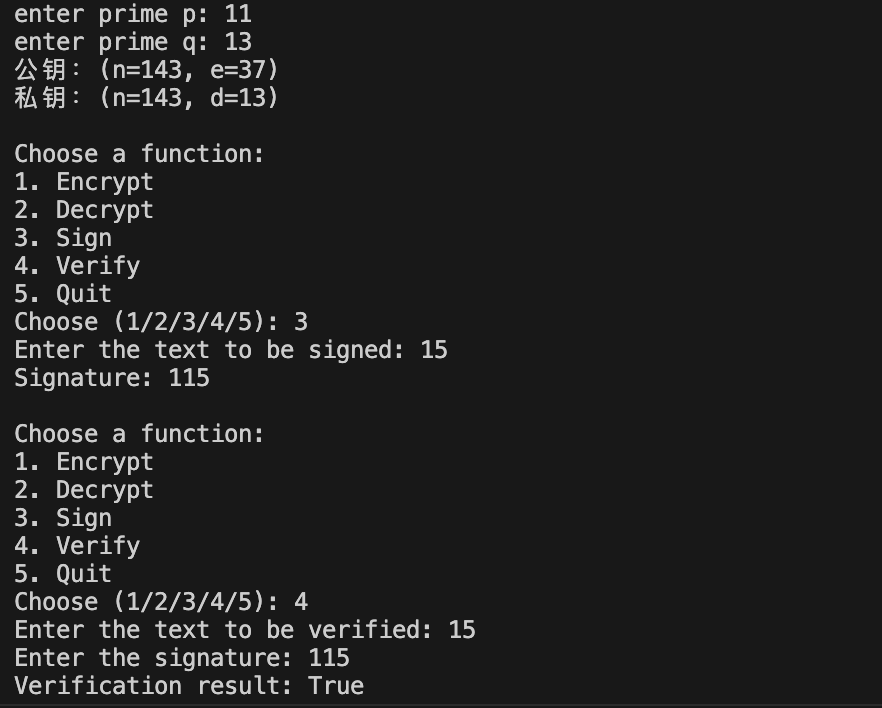


Рисунок 1 – Результат запкска программы

В ходе тестирования было обнаружено, что программа может подписывать и проверять только одну букву и не может работать с длинным текстом. На практике обычно выбирается модуль n соответствующего размера, чтобы число не переполнилось, превысив размер модуля. В коде используется значение e, равное 37, что является относительно небольшим значением, которое может привести к превышению числа над модулем n при подписании и проверке. Чтобы решить эту проблему, попробуйте использовать больший модуль n и убедитесь, что он находится в нужном диапазоне как для подписания, так и для проверки. Можно попробовать выбрать большие простые числа p и q для большего значения n. А также изменить хэш-функцию для повышения безопасности. Результат модификации приведен ниже:

import hashlib

def is\_prime(num):

if num < 2:

return False

for i in range(2, int(num\*\*0.5) + 1):

if num % i == 0:

return False

return True

def gcd(a, b):

while b:

a, b = b, a % b

return a

def mod\_inverse(a, m):

m0, x0, x1 = m, 0, 1

while a > 1:

q = a // m

m, a = a % m, m

x0, x1 = x1 - q \* x0, x0

return x1 + m0 if x1 < 0 else x1

def generate\_keypair(p, q):

if not (is\_prime(p) and is\_prime(q)):

raise ValueError("p and q must be primes")

n = p \* q

phi = (p - 1) \* (q - 1)

e = 65537

d = mod\_inverse(e, phi)

print(f"公钥：(n={n}, e={e})")

print(f"私钥：(n={n}, d={d})")

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

def encrypt(message, public\_key):

n, e = public\_key

cipher\_text = []

for char in message:

cipher\_text.append(pow(ord(char), e, n))

return cipher\_text, n

def decrypt(cipher\_text, private\_key, n):

\_, d = private\_key

decrypted\_message = ''

for char in cipher\_text:

decrypted\_message += chr(pow(char, d, n))

return decrypted\_message

def sign(message, private\_key, n):

\_, d = private\_key

hash\_value = hashlib.sha256(message.encode()).digest()

signature = pow(int.from\_bytes(hash\_value, byteorder="big"), d, n)

return signature

def verify(message, signature, public\_key, n):

hash\_value = hashlib.sha256(message.encode()).digest()

expected\_signature = pow(int.from\_bytes(hash\_value, byteorder="big"), public\_key[1], n)

return signature == expected\_signature

def main():

p = int(input("enter prime p: "))

q = int(input("enter prime q: "))

if not (is\_prime(p) and is\_prime(q)):

print("p and q must be primes.")

return

public\_key, private\_key = generate\_keypair(p, q)

while True:

print("\nChoose a function:")

print("1. Encrypt")

print("2. Decrypt")

print("3. Sign")

print("4. Verify")

print("5. Quit")

choice = input("Choose (1/2/3/4/5): ")

if choice == '1':

message = input("Enter the text to be encrypted: ")

cipher\_text, n = encrypt(message, public\_key)

print(f"Encryption results: {cipher\_text}, n={n}")

elif choice == '2':

cipher\_text = eval(input("Enter to be decrypted (as a list): "))

n = int(input("Enter n: "))

decrypted\_message = decrypt(cipher\_text, private\_key, n)

print(f"Decryption results: {decrypted\_message}")

elif choice == '3':

message = input("Enter the text to be signed: ")

signature = sign(message, private\_key, public\_key[0])

print(f"Signature: {signature}")

elif choice == '4':

message = input("Enter the text to be verified: ")

signature = int(input("Enter the signature: "))

result = verify(message, signature, public\_key, public\_key[0])

print(f"Verification result: {result}")

elif choice == '5':

break

else:

print("Invalid choice.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

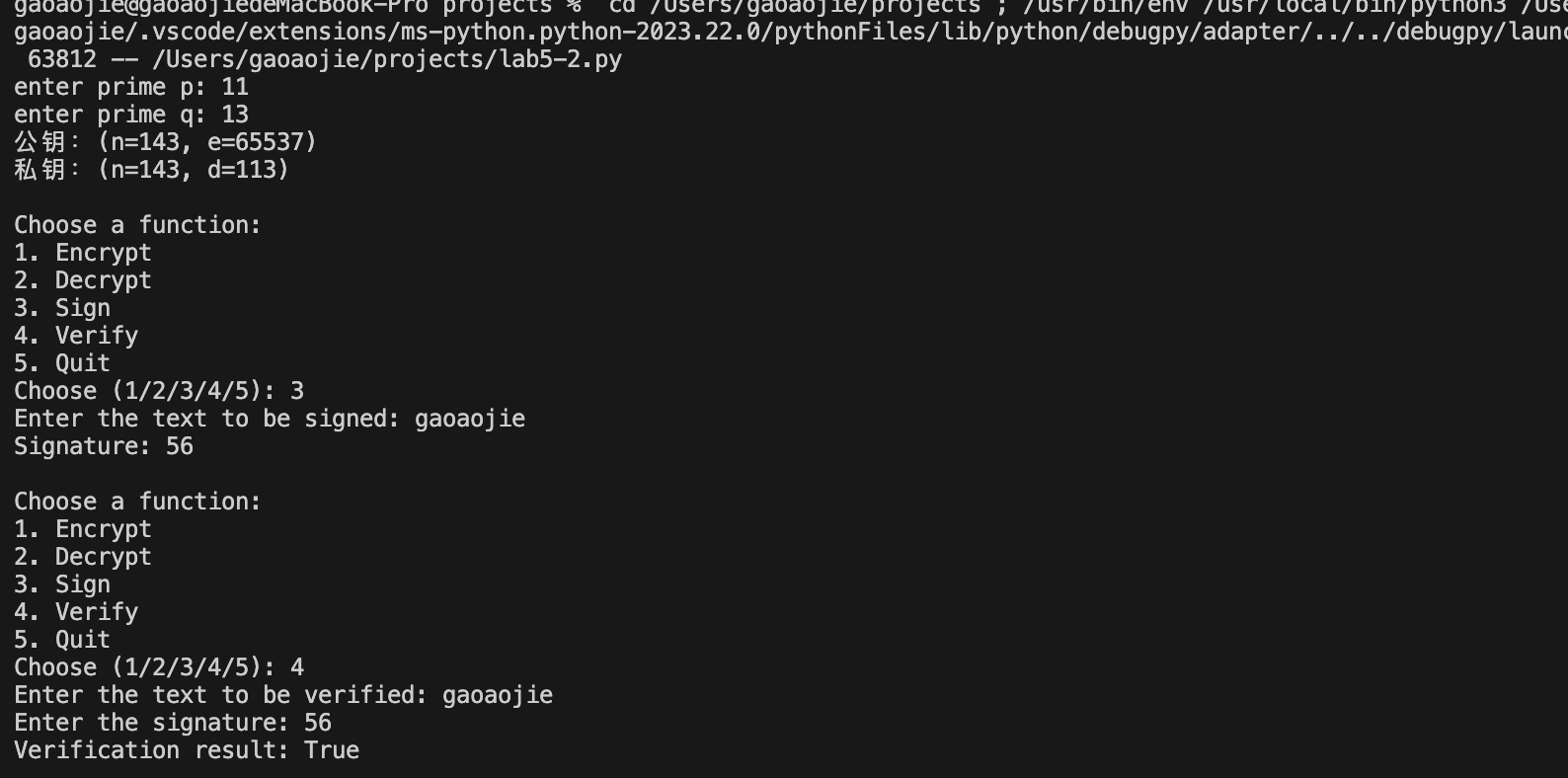


Рисунок 2 – Результат зашифрования и проверки

## Вывод

В данной работе завершена базовая реализация алгоритма RSA, разработана процедура вычисления и проверки ЭЦП. Она демонстрирует ключевые понятия RSA и позволяет выполнять попытки шифрования, дешифрования, создания цифровых подписей и аутентификации.

Следует отметить еще несколько моментов:

В коде используются базовые функции для проверки оригинальности и модульная арифметика. В реальных приложениях можно использовать более эффективную реализацию.

В коде не предусмотрена обработка ошибок, связанных с недействительными входными данными или потенциальными математическими ошибками.

В производственных приложениях рекомендуется использовать более надежные и безопасные реализации алгоритма RSA, например, входящие в состав криптографических библиотек.

Приложение

def is\_prime(num):

if num < 2:

return False

for i in range(2, int(num\*\*0.5) + 1):

if num % i == 0:

return False

return True

def gcd(a, b):

while b:

a, b = b, a % b

return a

def mod\_inverse(a, m):

m0, x0, x1 = m, 0, 1

while a > 1:

q = a // m

m, a = a % m, m

x0, x1 = x1 - q \* x0, x0

return x1 + m0 if x1 < 0 else x1

def generate\_keypair(p, q):

if not (is\_prime(p) and is\_prime(q)):

raise ValueError("p and q must be primes")

n = p \* q

phi = (p - 1) \* (q - 1)

e = 37

d = mod\_inverse(e, phi)

print(f"公钥：(n={n}, e={e})")

print(f"私钥：(n={n}, d={d})")

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

def encrypt(message, public\_key):

n, e = public\_key

cipher\_text = [pow(ord(char), e, n) for char in message]

return cipher\_text, n

def decrypt(cipher\_text, private\_key, n):

\_, d = private\_key

decrypted\_message = ''.join([chr(pow(char, d, n)) for char in cipher\_text])

return decrypted\_message

def sign(message, private\_key, n):

\_, d = private\_key

hash\_value = hash\_function(message)

signature = pow(hash\_value, d, n)

return signature

def verify(message, signature, public\_key, n):

n, e = public\_key

hash\_value = hash\_function(message)

hash\_from\_signature = pow(signature, e, n)

return hash\_value == hash\_from\_signature

def hash\_function(message):

return sum(ord(char) for char in message)

def main():

p = int(input("enter prime p: "))

q = int(input("enter prime q: "))

if not (is\_prime(p) and is\_prime(q)):

print("p and q must be primes.")

return

public\_key, private\_key = generate\_keypair(p, q)

while True:

print("\nChoose a function:")

print("1. Encrypt")

print("2. Decrypt")

print("3. Sign")

print("4. Verify")

print("5. Quit")

choice = input("Choose (1/2/3/4/5): ")

if choice == '1':

message = input("Enter the text to be encrypted: ")

cipher\_text, n = encrypt(message, public\_key)

print(f"Encryption results: {cipher\_text}, n={n}")

elif choice == '2':

cipher\_text = eval(input("Enter to be decrypted (as a list): "))

n = int(input("Enter n: "))

decrypted\_message = decrypt(cipher\_text, private\_key, n)

print(f"Decryption results: {decrypted\_message}")

elif choice == '3':

message = input("Enter the text to be signed: ")

signature = sign(message, private\_key, public\_key[0])

print(f"Signature: {signature}")

elif choice == '4':

message = input("Enter the text to be verified: ")

signature = int(input("Enter the signature: "))

result = verify(message, signature, public\_key, public\_key[0])

print(f"Verification result: {result}")

elif choice == '5':

break

else:

print("Invalid choice.")

if \_\_name\_\_ == "\_\_main\_\_":

main()