HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN

BÁO CÁO BÀI THỰC HÀNH
HỌC PHẦN: THỰC TẬP CƠ SỞ
MÃ HỌC PHẦN: INT13147

BÀI THỰC HÀNH 4.2
LẬP TRÌNH THUẬT TOÁN MẬT MÃ HỌC

Sinh viên thực hiện:

B22DCAT251 Đặng Đức Tài

Giảng viên hướng dẫn: TS. Phạm Hoàng Duy

HỌC KỲ 2 NĂM HỌC 2024-2025

MỤC LỤC

DANH MỤC CÁC HÌNH VẼ

CHƯƠNG 1. GIỚI THIỆU CHUNG VỀ BÀI THỰC HÀNH

1.1 Mục đích

- Sinh viên tìm hiểu một giải thuật mã hóa phổ biến và lập trình được chương trình mã hóa và giải mã sử dụng ngôn ngữ lập trình phổ biến như C/C++/Python/Java, đáp ứng chạy được với số lớn.

1.2 Tìm hiểu lý thuyết

1.2.1 Giới thiệu

- Báo cáo này trình bày lý thuyết cơ bản về lập trình xử lý số lớn với các phép toán cơ bản và giải thuật mật mã khóa công khai RSA. Mục tiêu là cung cấp kiến thức nền tảng để ứng dụng trong lập trình và bảo mật thông tin, giúp người đọc hiểu rõ cách thực hiện các phép toán với số lớn và cơ chế hoạt động của RSA.

1.2.2 Lập trình số lớn với các phép toán cơ bản

1.2.2.1 Khái niệm số lớn

- Trong lập trình, số lớn được hiểu là những số nguyên có kích thước vượt xa giới hạn của các kiểu dữ liệu nguyên chuẩn, chẳng hạn như kiểu int 32-bit (giới hạn khoảng ±2 tỷ) hoặc int 64-bit. Khi làm việc với các số có hàng trăm hoặc hàng nghìn chữ số, các ngôn ngữ lập trình thông thường không thể xử lý trực tiếp. Do đó, cần sử dụng các thư viện chuyên biệt hoặc tự thiết kế các cấu trúc dữ liệu để lưu trữ và thao tác trên những số này. Số lớn thường được biểu diễn dưới dạng chuỗi ký tự hoặc mảng các chữ số để dễ dàng xử lý.

1.2.2.2 Các phép toán cơ bản

- Để thực hiện các phép toán trên số lớn, các lập trình viên cần triển khai hoặc sử dụng các thuật toán phù hợp. Dưới đây là các phép toán cơ bản:

- Phép cộng và trừ: Hai phép toán này được thực hiện bằng cách xử lý từng chữ số từ phải sang trái, tương tự như cách tính tay. Trong phép cộng, cần chú ý đến số nhớ (carry) khi tổng của hai chữ số vượt quá 9. Trong phép trừ, số mượn (borrow) được sử dụng khi chữ số bị trừ nhỏ hơn chữ số trừ. Các thuật toán này đảm bảo tính chính xác ngay cả với số có kích thước lớn.
- Phép nhân: Phép nhân số lớn phức tạp hơn nhiều so với phép cộng và trừ. Phương pháp đơn giản nhất là thuật toán nhân trường học, trong đó mỗi chữ số của số này được nhân với từng chữ số của số kia, sau đó cộng các kết quả lại. Tuy nhiên, với số rất lớn, thuật toán Karatsuba được sử dụng để giảm độ phức tạp tính toán, chia bài toán thành các phần nhỏ hơn.
- Phép chia và modulo: Phép chia số lớn thường dựa trên thuật toán chia dài, tương tự cách chia tay. Ngoài ra, các phương pháp tối ưu như chia nhị phân có thể được áp dụng để tăng hiệu suất. Phép modulo (tính dư) cũng rất quan trọng, đặc biệt trong các ứng dụng mật mã như RSA.
- Phép lũy thừa: Tính lũy thừa của số lớn, chẳng hạn (a^b), là một bài toán tốn kém về mặt tính toán. Để tối ưu, thuật toán lũy thừa nhị phân (square-and-multiply) được sử

dụng, giảm số lần nhân xuống đáng kể bằng cách tận dụng biểu diễn nhị phân của số mũ.

1.2.2.3 Thư viện hỗ trợ

- Để đơn giản hóa việc xử lý số lớn, nhiều ngôn ngữ lập trình cung cấp các thư viện tích hợp sẵn. Ví dụ, trong Python, kiểu dữ liệu int hỗ trợ số nguyên không giới hạn kích thước, cho phép lập trình viên thực hiện các phép toán mà không cần lo lắng về tràn số. Trong Java, lớp BigInteger cung cấp các hàm để thực hiện tất cả các phép toán cơ bản trên số lớn. Đối với C++, thư viện GMP (GNU Multiple Precision Arithmetic Library) là một lựa chọn mạnh mẽ, hoặc lập trình viên có thể tự xây dựng cấu trúc dữ liệu để quản lý số lớn. Các thư viện này giúp tiết kiệm thời gian và đảm bảo độ chính xác.

1.2.3 Giải thuật mật mã khóa công khai RSA

1.2.3.1 Khái niệm

- RSA, được đặt theo tên ba nhà phát minh Rivest, Shamir và Adleman, là một trong những thuật toán mật mã khóa công khai đầu tiên và được sử dụng rộng rãi nhất. Khác với mật mã đối xứng, RSA sử dụng cặp khóa: khóa công khai để mã hóa dữ liệu và khóa bí mật để giải mã. Tính bảo mật của RSA dựa trên bài toán khó về phân tích thừa số nguyên tố, tức là khó xác định hai số nguyên tố lớn từ tích của chúng.

1.2.3.2 Nguyên lý hoạt động

- Quá trình hoạt động của RSA bao gồm ba bước chính: tạo khóa, mã hóa và giải mã.
- Tạo khóa:
- Đầu tiên, chọn hai số nguyên tố lớn (p) và (q), thường có kích thước hàng trăm chữ số để đảm bảo tính bảo mật.
- Tính modulus $n=p\cdot q$, đây là số được sử dụng trong cả mã hóa và giải mã.
- Tính giá trị hàm Euler $\varphi(n)=(p-1)(q-1)$, biểu thị số lượng số nguyên dương nhỏ hơn (n) và nguyên tố cùng nhau với (n).
- Chọn một số (e) (khóa công khai) sao cho $1<e<\varphi(n)$ và $\gcd(e,\varphi(n))=1$ (số e nguyên tố cùng nhau với $\varphi(n)$) .Thông thường, (e = 65537) là một lựa chọn phổ biến vì tính hiệu quả.
- Tính số (d) (khóa bí mật) sao cho $(d\cdot e \equiv 1 \pmod{\varphi(n)})$. Điều này được thực hiện bằng thuật toán Euclid mở rộng.
- Kết quả là cặp khóa công khai ((e, n)) và khóa bí mật ((d, n)).
- Mã hóa: Để mã hóa một thông điệp (m) (được biểu diễn dưới dạng số nguyên nhỏ hơn (n)), tính bản mã (c) theo công thức (c = m^e mod n). Phép lũy thừa và modulo này đòi hỏi xử lý số lớn hiệu quả.
- Giải mã: Để giải mã, từ bản mã (c), tính lại thông điệp gốc (m) bằng công thức (m = c^d mod n). Quá trình này cũng sử dụng các thuật toán xử lý số lớn.

1.2.3.3 Ứng dụng

- RSA có vai trò quan trọng trong nhiều lĩnh vực bảo mật:

- Bảo mật truyền thông: RSA được sử dụng trong các giao thức như SSL/TLS để bảo vệ dữ liệu truyền qua internet, chẳng hạn khi truy cập các trang web HTTPS.
- Chữ ký số: RSA cho phép tạo chữ ký số để xác thực danh tính người gửi và đảm bảo tính toàn vẹn của dữ liệu.
- Trao đổi khóa: RSA thường được dùng để trao đổi khóa an toàn cho các thuật toán mã hóa đối xứng như AES, vốn nhanh hơn trong việc mã hóa dữ liệu lớn.

1.2.3.4 Thách thức

- Mặc dù mạnh mẽ, RSA cũng đối mặt với một số thách thức:
- Hiệu suất: Các phép toán lũy thừa và modulo trên số lớn rất tốn kém về mặt tính toán, đặc biệt khi kích thước khóa lớn (thường là 2048 hoặc 4096 bit).
- Bảo mật: Độ an toàn của RSA phụ thuộc vào việc chọn các số nguyên tố lớn và ngẫu nhiên. Nếu (p) và (q) không được chọn cẩn thận, hoặc nếu có lỗ hổng trong việc tạo số ngẫu nhiên, hệ thống có thể bị tấn công.
- Tấn công: Các cuộc tấn công như phân tích thừa số (factoring) hoặc tấn công kênh bên (side-channel attacks) có thể làm suy yếu RSA nếu không được triển khai đúng cách. Ngoài ra, sự phát triển của máy tính lượng tử trong tương lai có thể đe dọa tính bảo mật của RSA.

CHƯƠNG 2. NỘI DUNG THỰC HÀNH

2.1 Chuẩn bị môi trường

- Phần mềm Visual Studio Code

2.2 Các bước thực hiện

2.2.1 Lập trình thư viện sử dụng trong giải thuật RSA

- Thư viện rsalib bao gồm các hàm xử lý sau:
- Hàm kiểm tra số nguyên tố sử dụng thuật toán Miller-Rabin

RSA > rsalib.py > ...

```python
import random
import math
# Miller-Rabin check prime number
def is_prime(n, k = 5):
    if n <= 1 or n == 4:
        return False
    if n <= 3:
        return True

    d = n - 1
    r = 0
    while d % 2 == 0:
        d //= 2
        r += 1

    for _ in range(k):
        a = random.randrange(2, n - 2)
        x = pow(a, d, n)

        if x == 1 or x == n - 1:
            continue

        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False

    return True
```

Hình 1 Hàm kiểm tra số nguyên tố

- Hàm sinh số nguyên tố ngẫu nhiên k bit

```python
# Gen prime number k (bits)
def gen_prime(bits):
    while True:
        candidate = random.getrandbits(bits) | (1 << bits - 1) | 1
        if is_prime(candidate):
            return candidate
```

Hình 2 Hàm sinh số nguyên tố ngẫu nhiên k bit

- Hàm tính modulo nghịch đảo

```python
# Calc mod invert in RSA
def mod_invert(e, phi):
    def extended_gcd(a, b):
        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

    gcd, x, _ = extended_gcd(e, phi)
    if gcd != 1:
        raise ValueError("Modular inverse does not exist")
    return (x % phi + phi) % phi
```

Hình 3 Hàm tính modulo nghịch đảo

- Hàm tính mũ

```python
# Calc pow
def mod_pow(base, exponent, modulus):
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent & 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent >>= 1
    return result
```

Hình 4 Hàm tính mũ

- Hàm chuyển đổi kiểu chuỗi sang kiểu số nguyên và ngược lại

```python
# Convert
def int2text(number):
    length = (number.bit_length() + 7) // 8
    return number.to_bytes(length, byteorder='big').decode('utf-8')

def text2int(text):
    return int.from_bytes(text.encode('utf-8'), byteorder='big')
```

Hình 5 Hàm chuyển đổi số sang chuỗi và ngược lại

- Hàm mã hóa / giải mã

```python
# Decrypt
def decrypt(c, d, n):
    return int2text(mod_pow(c, d, n))

# Encrypt
def encrypt(flag, e, n):
    m = text2int(flag)
    return mod_pow(m, e, n)
```

Hình 6 Hàm mã hóa / giải mã RSA

- Triển khai thư viện

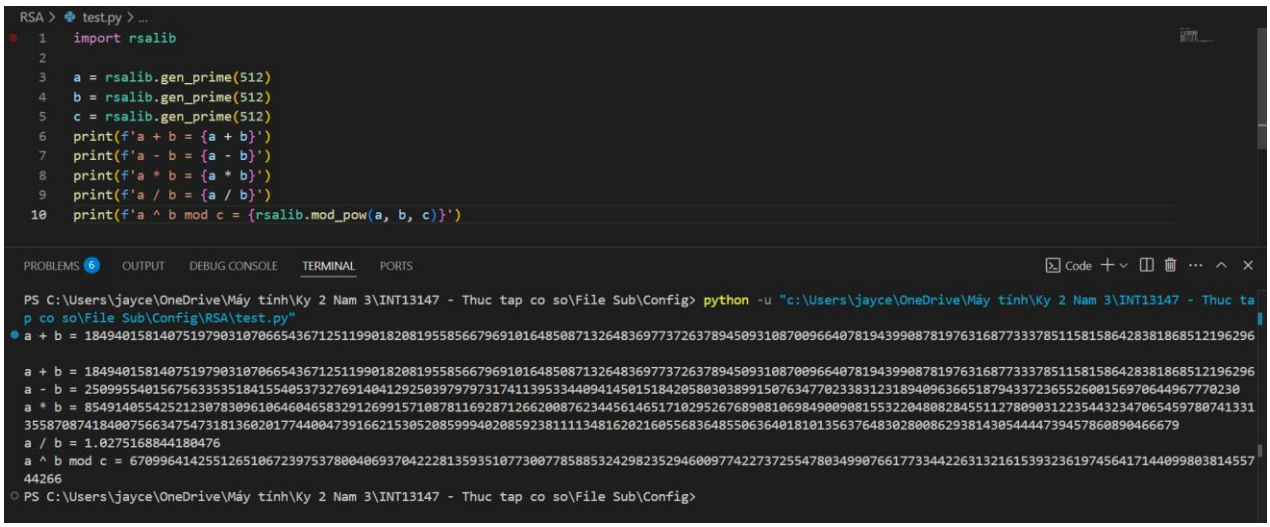
RSA > __init__.py

```
1  from .rsalib import *
```

Hình 7 Triển khai thư viện

2.2.2 Thử nghiệm

- Thử nghiệm với các phép tính toán số lớn



Hình 8 Thử nghiệm các phép toán số lớn

2.2.3 Lập trình mã hóa / giải mã & thử nghiệm với thông điệp

- Các bước thực hiện

- Thông điệp được mã hóa là “I am B22DCAT251” được đọc từ file
- Sinh ra cặp số nguyên tố (p, q) 512 bit
- Tính toán các giá trị cần thiết n, phi
- Mã hóa thông điệp c
- Giải mã thông điệp decrypted_flag

Hình 9 Code thử nghiệm mã hóa / giải mã

- Kết quả

Hình 10 Kết quả thực hiện

TÀI LIỆU THAM KHẢO

[1] https://cryptohack.org/