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:

B22DCAT199 Đỗ Duy Nam

Giảng viên hướng dẫn: TS.Đinh Trường 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 Lập trình số lớn về các phép toán cơ bản

Lập trình số lớn (arbitrary-precision arithmetic) là kỹ thuật xử lý các số có kích thước vượt quá giới hạn của kiểu dữ liệu nguyên chuẩn (như int, long trong các ngôn ngữ lập trình). Các số lớn thường được sử dụng trong các bài toán yêu cầu độ chính xác cao hoặc xử lý số có hàng trăm chữ số. Dưới đây là tổng quan về lập trình số lớn với các phép toán cơ bản (cộng, trừ, nhân, chia):

a) Tổng quan về số lớn

- **Vấn đề**: Các kiểu dữ liệu nguyên thông thường (như int32, int64) có giới hạn về kích thước (32-bit, 64-bit). Ví dụ, số lớn nhất mà int64 có thể lưu là khoảng 2^63 - 1 (~9.2 × 10^18).
- **Giải pháp**: Số lớn được biểu diễn dưới dạng chuỗi (string) hoặc mảng các chữ số, cho phép xử lý số có kích thước tùy ý.

b) Biểu diễn số lớn

- Số lớn thường được lưu dưới dạng:
    - **Chuỗi chữ số**: Ví dụ, số 12345678901234567890 được lưu là "12345678901234567890".
    - **Mảng chữ số**: Mỗi phần tử trong mảng chứa một chữ số hoặc một nhóm chữ số (ví dụ, lưu theo cơ số 10 hoặc 10^9 để tối ưu).
    - **Cơ số lớn**: Để tối ưu bộ nhớ và tốc độ, có thể lưu số theo cơ số lớn hơn (như 10^9 thay vì 10).
- **Lưu ý**:
    - Khi lưu dưới dạng chuỗi, cần xử lý ký tự (chuyển đổi giữa ký tự và số).
    - Khi lưu dưới dạng mảng, cần quản lý độ dài mảng và dấu của số (dương/âm).

c) Các phép toán cơ bản

- **Cộng số lớn**

- **Ý tưởng**: Thực hiện phép cộng giống như cách làm thủ công trên giấy (cộng từng cột, xử lý nhớ).
- **Thuật toán**:
    1. Đảo ngược chuỗi hoặc mảng chữ số để xử lý từ chữ số thấp đến cao.

2. Cộng từng cặp chữ số cùng vị trí, cộng thêm giá trị nhớ (carry) từ bước trước.
3. Lưu kết quả và cập nhật carry (nếu tổng ≥ 10).
4. Sau khi cộng xong, đảo ngược kết quả để trả về dạng đúng.

- **Trừ số lớn**

- **Ý tưởng**: Thực hiện phép trừ tương tự cách làm thủ công (trừ từng cột, xử lý mượn).
- **Thuật toán**:
    1. Đảm bảo số bị trừ lớn hơn số trừ (so sánh chuỗi hoặc độ dài).
    2. Đảo ngược chuỗi để xử lý từ chữ số thấp đến cao.
    3. Trừ từng cặp chữ số, mượn từ chữ số bên trái nếu cần.
    4. Loại bỏ các số 0 thừa ở đầu kết quả.

- **Nhân số lớn**

- **Ý tưởng**: Thực hiện phép nhân giống cách làm thủ công (nhân từng chữ số, cộng các tích).
- **Thuật toán**:
    1. Nhân từng chữ số của số thứ hai với toàn bộ số thứ nhất, dịch trái kết quả theo vị trí.
    2. Cộng tất cả các tích lại bằng phép cộng số lớn.
    3. Xử lý carry và loại bỏ số 0 thừa.

- **Chia số lớn**

- **Ý tưởng**: Thực hiện phép chia giống cách làm thủ công (chia từng nhóm chữ số, tìm thương và dư).
- **Thuật toán**:
    1. Lấy từng nhóm chữ số từ số bị chia.
    2. Tìm thương bằng cách thử hoặc chia nhị phân.
    3. Cập nhật dư và tiếp tục với các chữ số tiếp theo.

1.2.2 Giải thuật mã hóa công khai RSA

Thuật toán mã hóa *RSA* là một thuật toán mã hóa khóa công khai. Thuật toán *RSA* được xây dựng bởi các tác giả *Ron Rivest, Adi Shamir* và *Len Adleman* tại học viện *MIT* vào năm 1977, và ngày nay đang được sử dụng rộng rãi. Về mặt tổng quát thuật toán *RSA* là một phương pháp mã hóa theo khối. Trong đó thông điệp *M* và bản mã *C* là các số nguyên từ 0 đến 2_i với *i* số bit của khối. Kích thước thường dùng của *i* là 1024 bit. Thuật toán *RSA* sử dụng hàm một chiều là vấn đề phân tích một số thành thừa số nguyên tố và bài toán Logarith rời rạc.

Quá trình mã hóa:

Để thực hiện mã hóa và giải mã, thuật toán *RSA* dùng phép lũy thừa modulo của lý thuyết số. Các bước thực hiện như sau:

- Chọn hai số nguyên tố lớn *p* và *q* và tính $N = pq$. Cần chọn p và q sao cho:

$M < 2^{i-1} < N < 2^i$ với *i* là chiều dài bản rõ.

- Tính $\Phi(n) = (p - 1)(q - 1)$

- Tìm một số e sao cho: {e và $\Phi(n)$ là 2 số nguyên tố cùng nhau và $0 < e < \Phi(n)$}

- Tìm một số d sao cho: $e.d \equiv 1 \bmod \Phi(n)$ (hay: $d = e^{-1} \bmod \Phi(n)$)

(*d* là nghịch đảo của e trong phép modulo $\Phi(n)$)

- Chọn khóa công khai K_U là cặp *(e, N)*, khóa riêng K_R là cặp *(d, N)*

- Việc mã hóa thực hiện theo công thức:

- Theo phương án 1, mã hóa: $C = E(M, K_U) = M^e \bmod N$ (3.5)
- Theo phương án 2, mã hóa chứng thực: $C = E(M, K_R) = M^d \bmod N$ (3.6)

Quá trình giải mã: Việc giải mã thực hiện theo công thức:

- Theo phương án 1, giải mã: $M = D(C,K_R) = C^d \bmod N$ (3.7)
- Theo phương án 2, mã hóa chứng thực: $M = D(C,K_U) = C^e \bmod N$ (3.8)

->Thông điệp *M* có kích thước *i-1* bit, bản mã *C* có kích thước *i* bit.

Đánh giá:

- Độ an toàn: Độ an toàn của RSA phụ thuộc vào bài toán phân tích n thành p và q. Với n đủ lớn (ví dụ, 2048-bit hoặc 4096-bit), việc phân tích thừa số là bất khả thi với công nghệ hiện tại.
- Hiệu suất: RSA yêu cầu các phép toán số lớn (bình phương modulo, lũy thừa modulo) với số có kích thước hàng nghìn bit, dẫn đến chi phí tính toán cao.
- Ứng dụng thực tế: RSA được sử dụng rộng rãi trong các ứng dụng bảo mật như trao đổi khóa, chữ ký số, và mã hóa dữ liệu truyền trực tuyến. Được tích hợp vào nhiều giao thức mạng như HTTPS, SSH và SSL/TLS.

Tóm lại, RSA là một giải thuật mã hóa mạnh mẽ, đáng tin cậy và được sử dụng rộng rãi trong hơn 40 năm qua. Tuy nhiên, với sự phát triển của công nghệ, đặc biệt là máy tính lượng tử, RSA đang đối mặt với các thách thức về bảo mật và hiệu suất. Để đảm bảo an toàn, cần sử dụng RSA đúng cách (khóa lớn, đệm an toàn) và xem xét chuyển đổi sang các thuật toán hiện đại hơn như ECC hoặc hậu lượng tử trong tương lai.

1.3 Kết chương

Ở chương này đã tìm hiểu về lập trình số lớn và các phép toán cơ bản liên quan đến xử lý số nguyên lớn. Bên cạnh đó cũng đã tìm hiểu về giải thuật mã hóa công khai RSA.

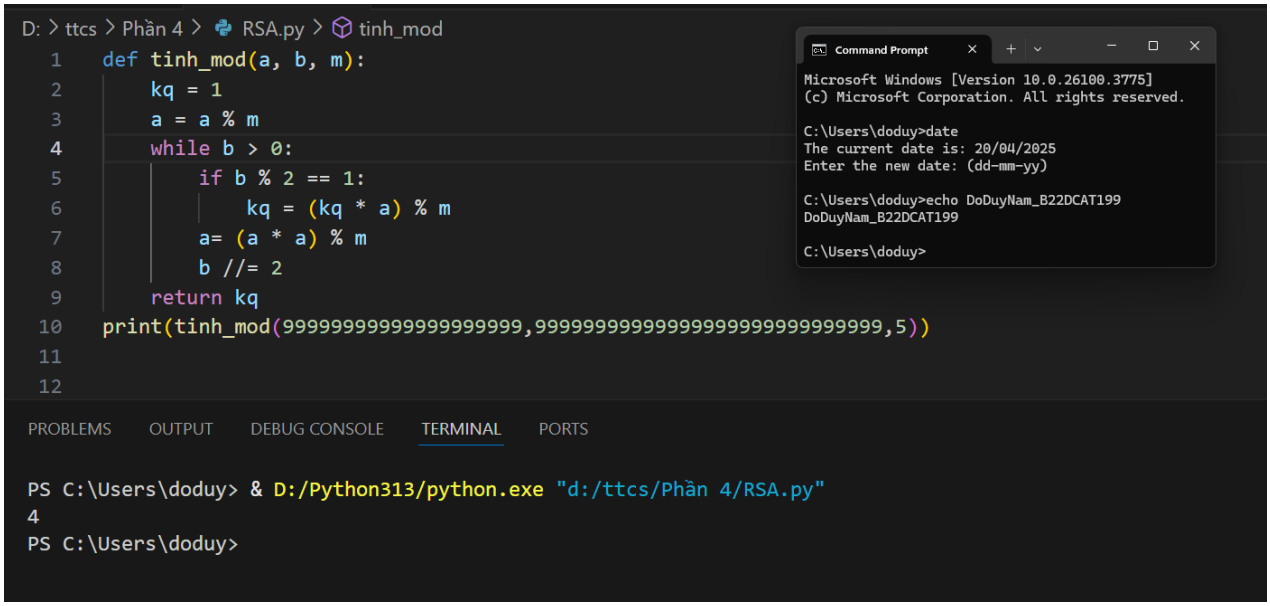
CHƯƠNG 2. NỘI DUNG THỰC HÀNH

2.1 Chuẩn bị môi trường

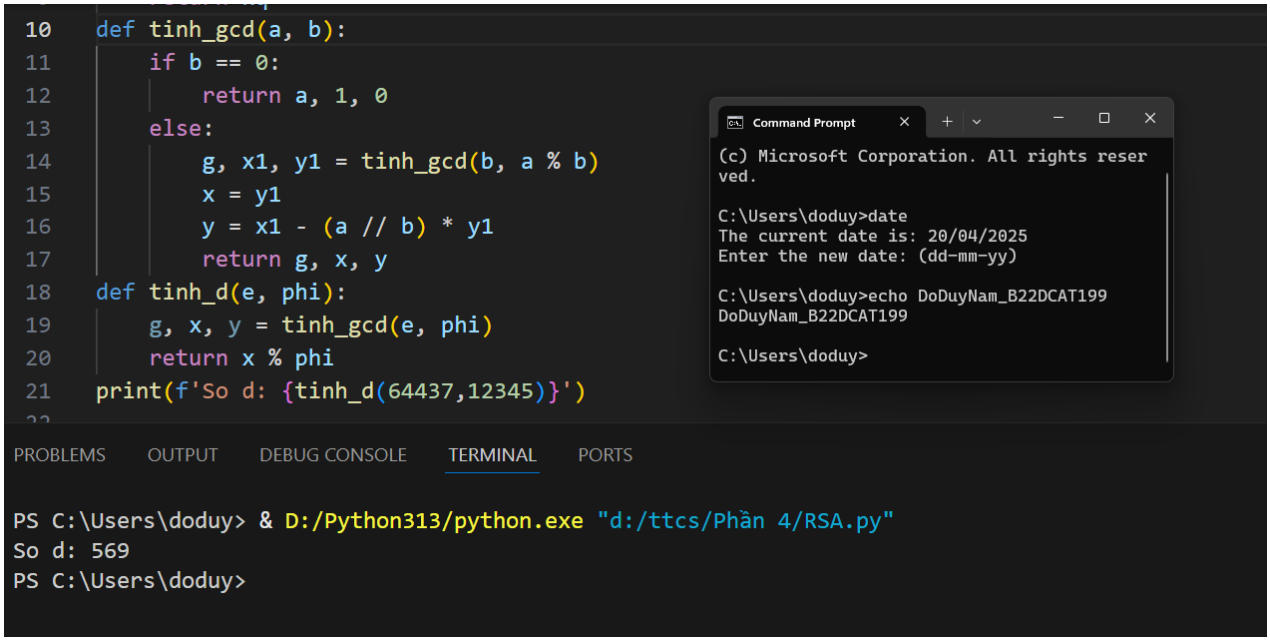
- Môi trường lập trình theo mong muốn. (Pycharm, VSCode,…)

2.2 Các bước thực hiện

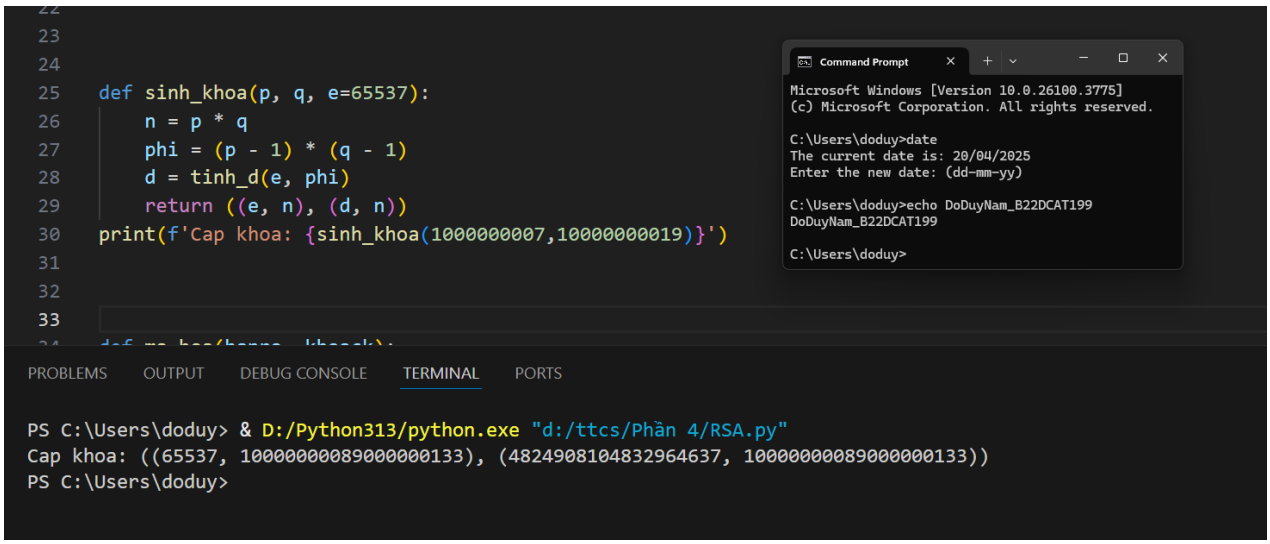
- Lập trình thư viện số lớn với các phép toán cơ bản để sử dụng trong giải thuật mã hóa/giải mã RSA
- Thử nghiệm chứng minh thư viện hoạt động tốt với các ví dụ phép toán cho số lớn
- Lập trình giải thuật mã hóa và giải mã
- Thử nghiệm mã hóa và giải mã chuỗi ký tự: “I am B22DCAT199”



Hình 1 Hàm tính a^b mod m

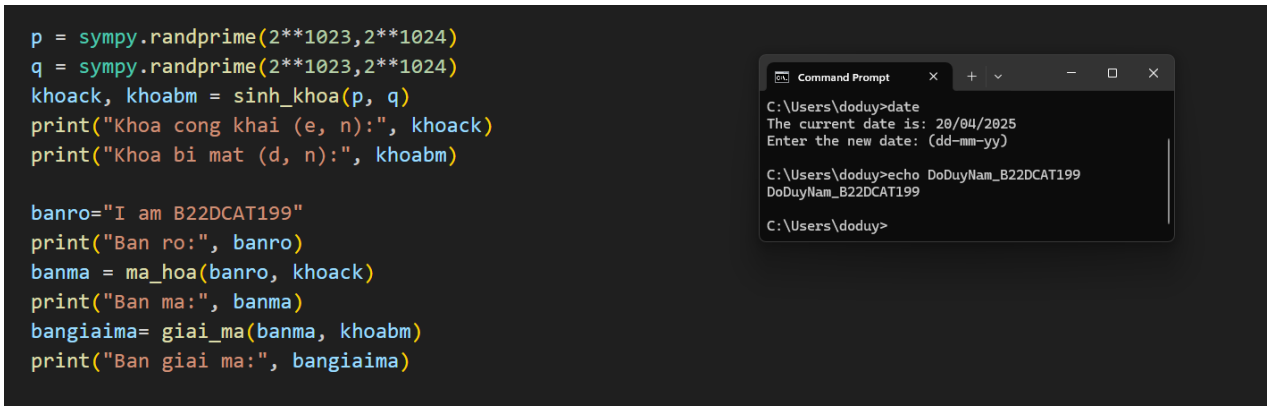


Hình 2 Hàm tính số d sao cho d.e mod phi=1

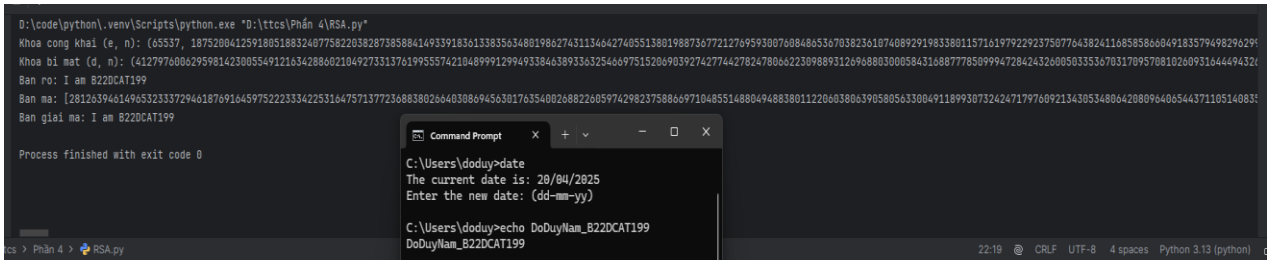


Hình 3 Hàm sinh khóa

Hình 4 Hàm mã hóa và giải mã



Hình 5 Thử nghiệm code với bản rõ là I am B22DCAT199



Hình 6 Kết quả nhận được

2.3 Kết chương

Ở chương này đã lập trình và thử nghiệm thành công các thư viện số lớn với các phép toán cơ bản để sử dụng trong giải thuật mã hóa/giải mã RSA. Dựa vào đó cũng đã lập trình thành công giải thuật mã hóa và giải mã RSA. Bên cạnh đó cũng đã thử nghiệm thành công việc mã hóa cũng như giải mã với một bản rõ cho trước.

KẾT LUẬN

- Tìm hiểu về lập trình số lớn với các phép toán cơ bản.
- Tìm hiểu về giải thuật mã hóa công khai RSA.
- Lập trình và thử nghiệm thành công thư viện số lớn với các phép toán cơ bản để sử dụng trong giải thuật mã hóa/giải mã RSA.
- Lập trình thành công giải thuật mã hóa và giải mã RSA.
- Thử nghiệm thành công mã hóa và giải mã chuỗi ký tự cho trước.

TÀI LIỆU THAM KHẢO

[1] Đỗ Xuân Chợ, Bài giảng Mật mã học cơ sở, Học Viện Công Nghệ Bưu Chính Viễn Thông, 2021.