

**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:

**B22DCAT206 PHẠM ĐỨC NAM**

Giảng viên hướng dẫn: **Đỗ Xuân Chợt**

**HỌC KỲ 2 NĂM HỌC 2024-2025**

## Mục lục

<b>1.</b>	Mục đích.....	3
<b>2.</b>	Nội dung thực hành .....	3
<b>2.1.</b>	Cơ sở lý thuyết .....	3
2.1.1.	Tìm hiểu về lập trình số lớn với các phép toán cơ bản .....	3
2.1.2.	Tìm hiểu về giải thuật mật mã khóa công khai RSA .....	4
<b>2.2.</b>	Các bước thực hiện.....	6
2.2.1.	Chuẩn bị môi trường .....	6
2.2.2.	Thực hiện.....	6
<b>3.</b>	Kết luận .....	10
<b>4.</b>	Tài liệu tham khảo .....	10

## Danh mục hình ảnh

Hàm tính $ab \bmod m$ .....	7
Hàm tìm số nghịch đảo $d$ sao cho $d.e \bmod m = 1$ .....	8
Hàm sinh khóa, để đơn giản ta chọn luôn $e = 65537$ .....	8
Hàm mã hóa và giải mã.....	9
Kết quả.....	9

## 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. Qua đó áp dụng nội dung kiến thức mình đã học vào thực tế

## 2. Nội dung thực hành

### 2.1. Cơ sở lý thuyết

#### 2.1.1. Tìm hiểu về lập trình số lớn với các phép toán cơ bản

Một số lý thuyết cơ bản về lập trình số nguyên lớn:

Lập trình số lớn (large number arithmetic) là một lĩnh vực quan trọng trong khoa học máy tính, đặc biệt khi xử lý các số nguyên vượt quá giới hạn kích thước của các kiểu dữ liệu nguyên thủy (như int hoặc long) trong các ngôn ngữ lập trình. Các số nguyên lớn thường xuất hiện trong các ứng dụng bảo mật, chẳng hạn như mã hóa RSA, nơi các số có độ dài hàng trăm hoặc hàng nghìn bit được sử dụng.

#### – Cộng và Trừ:

- + Các phép toán cộng và trừ có thể được thực hiện bằng cách duyệt qua từng chữ số của hai số và thực hiện phép toán tương ứng từng cặp chữ số, kèm theo việc xử lý nhớ và mượn nếu cần.
- + Bắt đầu từ chữ số đơn vị và tiếp tục lần lượt đến chữ số hàng đơn vị, thực hiện phép toán cộng hoặc trừ tương ứng.

#### – Nhân:

- + Phép nhân hai số nguyên lớn có thể thực hiện bằng thuật toán nhân dài (long multiplication) hoặc các thuật toán nhân nhanh hơn như nhân Karatsuba hoặc nhân Toom-Cook.
- + Thuật toán nhân dài thực hiện phép nhân tương tự như cách thực hiện phép nhân trong toán học cơ bản: duyệt qua từng chữ số của một số và nhân với từng chữ số của số kia, sau đó cộng các kết quả lại.

#### – Chia:

- + Phép chia hai số nguyên lớn có thể thực hiện bằng thuật toán chia dài (long division) hoặc các thuật toán chia hiệu quả hơn như thuật toán chia như thuật toán chia nhân bán (divide-and-conquer division) hoặc thuật toán chia nhưng tốt hơn (Newton-Raphson division).
- + Tương tự như phép nhân, thuật toán chia dài duyệt qua từng chữ số của số chia và thực hiện phép chia tương ứng.

## – Tính Lũy Thừa:

- + Phép tính lũy thừa của một số nguyên lớn có thể được thực hiện bằng thuật toán lũy thừa nhanh. Thuật toán này giảm thiểu số lượng phép toán bằng cách phân tích số mũ thành các phần nhỏ và tính lũy thừa của từng phần.

## – Tìm Ước Chung Lớn Nhất (GCD):

- + Để tìm ước chung lớn nhất của hai số lớn, các thuật toán như **thuật toán Euclid mở rộng** (extended Euclidean algorithm) hoặc **thuật toán nhị phân GCD** (binary GCD algorithm) được sử dụng.
- + Thuật toán Euclid dựa trên việc lặp lại phép chia lấy nguyên và lấy dư cho đến khi phần dư bằng 0. Thuật toán mở rộng còn cung cấp các hệ số Bézout, giúp tìm nghịch đảo modulo – một bước quan trọng trong RSA.
- + Thuật toán nhị phân GCD tối ưu hơn cho số lớn vì nó sử dụng các phép toán bit thay vì phép chia, giảm chi phí tính toán.

Trong lập trình số nguyên lớn, việc tối ưu hóa và cải thiện hiệu suất là rất quan trọng. Điều này bao gồm việc sử dụng các thuật toán hiệu quả nhất để thực hiện các phép toán cơ bản và giải quyết các vấn đề liên quan đến hiệu năng và bộ nhớ.

Trong Python, số nguyên không bị giới hạn về giá trị. Tuy nhiên, số nguyên có thể lớn đến mức nào phụ thuộc vào bộ nhớ có sẵn trên hệ thống mà bạn đang sử dụng. Chính vì vậy, ta có thể xử lý số nguyên lớn dễ dàng với Python.

### 2.1.2. Tìm hiểu về giải thuật mật mã khóa công khai RSA

#### a. Giải thuật mã hóa khóa công khai

**Mã hóa khóa bất đối xứng** (Hay còn gọi là mã hóa khóa công khai): là phương pháp mã hóa mà khóa sử dụng để mã hóa sẽ khác với khóa dùng để giải mã. Khóa dùng để mã hóa gọi là khóa công khai và khóa dùng để giải mã gọi là khóa bí mật. Tất cả mọi người đều có thể biết được khóa công khai (kể cả hacker), và có thể dùng khóa công khai để mã hóa thông tin. Nhưng chỉ có người nhận mới nắm giữ khóa bí mật, nên chỉ có người nhận mới có thể giải mã được thông tin.

**Đặc điểm nổi bật** của các hệ mã hóa khóa bất đối xứng là kích thước khóa lớn, lên đến hàng ngàn bit. Do vậy, các hệ mã hóa dạng này thường có tốc độ thực thi chậm hơn nhiều lần so với các hệ mã hóa khóa đối xứng với độ an toàn tương đương. Mặc dù vậy, các hệ mã hóa khóa bất đối xứng có khả năng đạt độ an toàn cao và ưu điểm nổi bật nhất là việc quản lý và phân phối khóa đơn giản hơn do khóa công khai có thể phân phối rộng rãi.

**Hệ thống mật mã hóa khóa công khai có thể sử dụng với các mục đích:**

- Mã hóa: giữ bí mật thông tin và chỉ có người có khóa bí mật mới giải mã được.

- Tạo chữ ký số: cho phép kiểm tra văn bản có phải đã được tạo với một khóa bí mật nào đó hay không.
- Thỏa thuận khóa: cho phép thiết lập khóa dung để trao đổi thông tin giữa hai bên.

**Các giải thuật mã hóa khóa bất đối xứng điển hình** bao gồm: RSA, Rabin, ElGamal, McEliece và Knapsack. Trong bài tiểu luận này, chúng tôi tìm hiểu và trình bày về giải thuật mã hóa RSA – một trong các giải thuật mã hóa khóa đối xứng được sử dụng rộng rãi nhất trên thực tế.

## b. Giải thuật mã hóa khóa 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.

### ❖ Thuật toán

Để thực hiện mã hóa và giải mã, thuật toán RSA dùng phép lũy thừa modulo của lý thuyết số.

### Quá trình sinh khóa

- 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:  $\{e \text{ và } \Phi(n) \text{ là 2 số nguyên tố cùng nhau và } 0 < e < \Phi(n)\}$
- Tìm một số  $d$  sao cho:  $e \cdot d \equiv 1 \pmod{\Phi(n)}$  (hay:  $d = e^{-1} \pmod{\Phi(n)}$ )  
( $d$  là nghịch đảo của  $e$  trong phép modulo  $\Phi(n)$ )
- Chọn khóa công khai  $Ku$  là cặp  $(e, N)$ , khóa riêng  $Kr$  là cặp  $(d, N)$

### Quá trình mã hóa:

- Thông điệp  $m$  được chuyển thành số:  $m < n$
- Bản mã  $c = m^e \pmod{n}$

### Quá trình giải mã:

- Bản mã  $c$ ,  $c < n$
- Bản rõ  $m = c^d \pmod{n}$

**Lưu ý:** Khi chọn hai số nguyên tố  $p$  và  $q$  để sử dụng trong hệ mã hoá RSA, quan trọng

nhất là chúng không nên quá gần nhau hoặc quá lệch nhau.

### ❖ **Đánh giá:**

#### **An toàn và bảo mật:**

- RSA dựa trên bài toán phân tích thừa số nguyên tố, được coi là khó với các thuật toán hiện tại. Tuy nhiên, chưa có bằng chứng toán học chứng minh rằng bài toán này không thể giải được trong tương lai.
- Với các khóa dài (2048 bit hoặc lớn hơn), RSA cung cấp mức độ bảo mật cao, đủ để chống lại các cuộc tấn công phổ biến.

#### **Khả năng chống tấn công:**

- RSA có thể chống lại các cuộc tấn công như brute force (dò khóa), man-in-the-middle (tấn công miền trung gian), và factorization (phân tích thừa số). Tuy nhiên, nếu sử dụng khóa ngắn (dưới 1024 bit), RSA có thể dễ bị tấn công bởi các thuật toán phân tích thừa số hiện đại như Quadratic Sieve hoặc General Number Field Sieve.

#### **Hiệu suất:**

- RSA có tốc độ xử lý chậm hơn so với các thuật toán mã hóa đối xứng (như AES) do yêu cầu tính toán các phép lũy thừa và modulo trên số lớn.
- Để cải thiện hiệu suất, RSA thường được sử dụng kết hợp với mã hóa đối xứng, trong đó RSA mã hóa khóa đối xứng, còn dữ liệu chính được mã hóa bằng thuật toán đối xứng nhanh hơn.

#### **Ứng dụng thực tế:**

- RSA được sử dụng rộng rãi trong các giao thức bảo mật như HTTPS, SSH, và SSL/TLS để mã hóa dữ liệu, ký số, và thỏa thuận khóa.
- RSA cũng được dùng trong các hệ thống chữ ký số để xác minh tính xác thực và toàn vẹn của thông điệp.

#### **Quản lý khóa:**

- Việc quản lý và bảo vệ khóa bí mật là một thách thức lớn. Nếu khóa bí mật bị rò rỉ, toàn bộ hệ thống bảo mật sẽ bị đe dọa.
- Khóa công khai có thể được phân phối công khai, giúp đơn giản hóa việc quản lý khóa so với mã hóa đối xứng.

#### **Chi phí tính toán:**

- RSA yêu cầu tài nguyên tính toán lớn, đặc biệt với các phép lũy thừa modulo trên số lớn. Điều này làm tăng chi phí tính toán, đặc biệt trong các hệ thống có tài nguyên hạn

chế (như thiết bị nhúng).

Tóm lại, RSA là một giải thuật mã hoá công khai mạnh mẽ và phổ biến, nhưng cũng cần cân nhắc về hiệu suất tính toán và quản lý khóa khi triển khai trong các ứng dụng thực tế.

## 2.2. Các bước thực hiện

### 2.2.1. Chuẩn bị môi trường

- Python 3.11.7

### 2.2.2. 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 B22DCAT206” (thay bằng mã sinh viên của mình vào)



```
27
28 def modInverse(a, m):
29     m0, x0, y0 = m, 1, 0
30     while a < m:
31         q = a // m
32         m, a, x0, y0 = a % m, m, x0 - q * y0, y0
33     if x0 < 0:
34         x0 += m
35     return x0
36
37
38 def pow_mod(base, exponent, modulus):
39     result = 1
40     base %= modulus
41     while exponent > 0:
42         if exponent % 2 == 1:
43             result = (result * base) % modulus
44             exponent >>= 1
45             base = (base * base) % modulus
46     return result
47
```

Hàm tính  $a^b \bmod m$

```

27
28 def modInverse(a, m):
29     m0, x0, x1 = m, 0, 1
30     while a > 1:
31         q = a // m
32         m, a = a % m, m
33         x0, x1 = x1 - q * x0, x0
34     if x1 < 0:
35         x1 += m0
36     return x1
37
38
39
40
41
42 B22DCAT206PHẠM ĐỨC NAM
43
44
45
46
47
48 # --- Tạo khóa RSA ---

```

Hàm tìm số nghịch đảo  $d$  sao cho  $d.e \bmod m = 1$

```

39
40 # 1, Sinh p và q ngẫu nhiên
41 p = generate_prime()
42 q = generate_prime()
43 # đảm bảo p và q khác nhau
44 while p == q:
45     q = generate_prime()
46
47 print(f">50 nguyên tố p = {p}")
48 print(f">50 nguyên tố q = {q}")
49
50 # 2, Tính n và phi(n)
51 n = p * q
52 phi = (p - 1) * (q - 1)
53 print(f">Modulus n = p * q = {n}")
54 print(f">Phi(n) = (p-1)*(q-1) = {phi}")
55
56 # 3, Chọn số mũ công khai e
57 e thường là một số nguyên tố nhỏ, phổ biến là 65537
58 e = 65537
59 # kiểm tra nếu gcd(e, phi) != 1 hoặc e == phi, chọn e
60 if gcd(e, phi) != 1 or e == phi:
61     # tìm e khác nhỏ nhất > 1 và nguyên tố cùng nhau
62     e = 2
63     while gcd(e, phi) != 1:
64         e += 2
65     # chỉ cần kiểm tra các số lẻ > 2

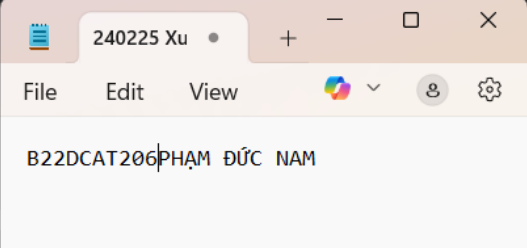
```

Hàm sinh khóa, để đơn giản ta chọn luôn  $e = 65537$

```

6
7 # --- Giải mã thông điệp ---
8
9 # 1. Nhập thông điệp đã mã hoá
10 # Cảnh báo: Không có kiểm tra lỗi, nếu nhập sai định d
11 encrypted_message_str = input("Nhập thông điệp đã mã h
12 # Chuyển đổi chuỗi nhập vào thành một list các số nguy
13 encrypted_message = ast.literal_eval(encrypted_message
14
15 # 2. Nhập khoá bí mật (n và d)
16 # Cảnh báo: Không có kiểm tra lỗi, nếu nhập không phải
17 n_str = input("Nhập giá trị n (modulus): ")
18 n = int(n_str)
19 d_str = input("Nhập giá trị d (private exponent): ")
20 d = int(d_str)
21
22 # 3. Giải mã từng số
23 decrypted_codes = [pow_mod(c, d, n) for c in encrypted
24
25 # 4. Chuyển đổi mã số thành ký tự
26 # Cảnh báo: Không có kiểm tra lỗi, nếu mã giải mã không
27 decrypted_message = "".join([chr(code) for code in dec
28
29
30 print("-" * 20)
31 print("--- KẾT QUẢ GIẢI MÃ ---")
32 print(f"Thông điệp mã hoá đã nhập: {encrypted_message}")
33 print(f"Khoá bí mật sử dụng: (n={n}, d={d})")
34 print(f"Thông điệp đã giải mã: {decrypted_message}")
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```



*Hàm mã hóa và giải mã*

```

-----
Nhập thông điệp cần mã hoá: HELLO
Thông điệp dạng số (ASCII): [72, 69, 76, 76, 79]
-----
--- KẾT QUẢ MÃ HÓA ---
Các tham số RSA:
p = 974230247
q = 489150967
n (modulus) = 476548642386500849
phi(n) = 476548600843113636
e (public exponent) = 65537
d (private exponent) = 192482398311406361
Thông điệp gốc: HELLO
Thông điệp đã mã hoá : [65637435911802090, 430887473021168276, 267979801516161941, 267979801516161941, 99284682109
© PS C:\Users\Admin\SimpleRSA>

```



*Kết quả*

### 3. Kết luận

- Lý thuyết về mã hóa khóa công khai
- Chạy giải thuật mã hóa RSA thành công

### 4. Tài liệu tham khảo

- [1]. Đỗ Xuân Chợt, Bài giảng Mật mã học cơ sở, Học viện Công Nghệ Bưu Chính Viễn Thông, 2021.
- [2]. Đỗ Xuân Chợt, Bài giảng Mật mã học nâng cao, Học viện Công Nghệ Bưu Chính Viễn Thông, 2021.