TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC NGHIỆM

Học phần: An toàn & Bảo mật thông tin

Chủ đề: Xây dựng chương trình mã hóa và giải mã RSA

Giáo viên hướng dẫn : TS. Phạm Văn Hiệp

Nhóm sinh viên thực hiện :

1. Ngô Trường Công Mã SV: 2021602766
2. Nguyễn Bá Tuấn Mã SV: 2021606840
3. Trần Đình Đức Mã SV: 2021602724
4. Phạm Phú Mỹ Mã SV: 2021601300

Mã Lớp học phần: 20241IT6001002

Nhóm: 03

Hà Nội - Năm 2024

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

LỜI NÓI ĐẦU

Ngay từ thuở sơ khai của lịch sử, vấn đề bảo mật và mã hóa thông tin đã tồn tại và được nghiên cứu suốt chiều dài văn minh nhân loại. Từ xa xưa, trước cả sự tồn tại của những khái niệm về máy tính sơ khai nhất, con người đã sáng tạo ra các hệ mật mã cổ điển với bút và giấy, thậm chí là hỗ trợ từ những dụng cụ cơ khí đơn giản.

Vào đầu thế kỷ 20, sự xuất hiện của các cơ cấu cơ khí và điện cơ, chẳng hạn như máy Enigma, đã cung cấp những cơ chế phức tạp và hiệu quả hơn cho việc mật mã hóa. Sự ra đời và phát triển mạnh mẽ của ngành điện tử và máy tính, cũng như vai trò ngày càng quan trọng của chúng trong đời sống cũng như nền văn minh làm bùng nổ tầm quan trọng của việc bảo mật và mã hóa thông tin, nhờ đó tạo điều kiện để mật mã học có một bước nhảy vọt lớn. Nhờ sự trợ giúp của máy tính, rất nhiều hệ mật mã hiện đại đã ra đời dựa trên cơ sở đại số Modulo và các thuật toán logarit rời rạc, đều là những hệ mật mã có tính bảo mật cao vượt trội. Một trong số các hệ mật mã hiện đại phổ biến nhất vẫn được sử dụng cho đến ngày hôm nay chính là hệ mã hóa RSA.

Dựa trên sự hướng dẫn của giảng viên – thầy Phạm Văn Hiệp, các thành viên trong nhóm đã nhất trí quyết định tiến hành tìm hiểu về các thuật toán mã hóa và giải mã RSA. Bên cạnh đó nhóm chúng em đã tiến hành xây dựng các chương trình demo hệ mã hóa và giải mã RSA bằng hai ngôn ngữ chính là C++ và C#.

Nội dung chính của bài báo cáo này bao gồm các cơ sở toán học cơ bản và nội dung của thuật toán trong RSA, đồng thời đưa ra những thiết kế xây dựng chương trình demo ứng dụng thuật toán RSA bằng hai ngôn ngữ lập trình C++ và C# để thầy và những người quan tâm có được cái nhìn tổng thể về đề tài của chúng e

CHƯƠNG 1 TỔNG QUAN

1.1 Tổng quan về An toàn bảo mật thông tin

An toàn thông tin là bảo vệ các đặc tính riêng tư (confidentialy), toàn vẹn (intergrity) và khả dụng (availabity) của thông tin.

- C: (Confidentialy) bảo vệ tính riêng tư của dữ liệu thông qua các cơ chế chứng thực và mã hóa, ngăn ngừa những người không hợp lệ sẽ không được ọc những thông tin. Giống như các bì thư khi phát lương thưởng được dán chữ Confidentialy, chúng ta có thể hình dung trong môi trường công nghệ thông tin là một người chưa đăng nhập vào Domain sẽ không được truy cập những dữ liệu chỉ chia sẻ cho các Domain User.

- I: (Intergrity) bảo vệ tính toàn vẹn của dữ liệu thông qua các thuật toán RSA, SHA, MD5 ... ngăn ngừa attacker thay đổi các thông tin nhạy cảm trong quá trình truyền.

- A: (Available) bảo đảm dữ liệu luôn ở trong trạng thái sẵn sằng đáp ứng nhu cầu của người dùng.

- Non-Repudiation: Tính không thể chối bỏ, nghĩa là dữ liệu người nào gửi đi thì họ phải có trách nhiệm với các thông tin của mình thông qua các xác nhận nguồn gốc như chữ kí điện tử.

- Để đảm bảo việc truyền tin an toàn và kiểm tra tính toàn vẹn của thông tin, người ta thường mã hóa thông tin trước khi truyền đi bằng các một sổ các hệ mật như DES, Triple DES (3DES), RC4, AES, RSA, Rabin, Diffle-Hellman, RSA,...

- Qua dự án lần này nhóm sinh viên chúng em đã đi sâu tìm hiểu về xây dựng và phát triển mã hóa giải mã RSA , nắm được những kiến thức cơ

bản trong phương pháp mã hóa và giải mã cũng như là các thuật toán cần thiết bằng các ngôn ngữ như: Java, Python, C++, PHP...

1.1.1 Vai trò của hệ mật mã

Phải che giấu được nội dung của văn bản rõ (plaintext)

Tạo các yếu tố xác thực thông tin, đảm bảo thông tin lưu hành trong hệ thống đến người nhận hợp pháp là xác thực.

1.1.2 Khái niệm cơ bản

- Bản rõ X: được gọi là bản tin gốc, bản rõ có thể được chia nhỏ có kích thước phù hợp.
- Bản mã Y: là bản tin gốc đã được mã hoá.
- Mã: là thuật toán mã hoá chuyển bản rõ thành bản mã, thông thường chúng ta cần thuật toán mã hoá mạnh, cho dù kẻ thù biết được thuật toán, nhưng không biết thông tin về khoá thì cũng không tìm được bản rỏ.
- Khoá K: là thông tin tham số dùng để mã hoá, chỉ có người gửi và người nhận biết. Khoá là độc lập với bản rõ và có độ dài phù hợp với yêu cầu bảo mật.
- Mã hoá: là quá trình chuyển bản rõ thành bản mã.
- Giải mã: là quá trình chuyển bản mã thành bản rõ, đây là quá trình ngược lại của mã hoá.

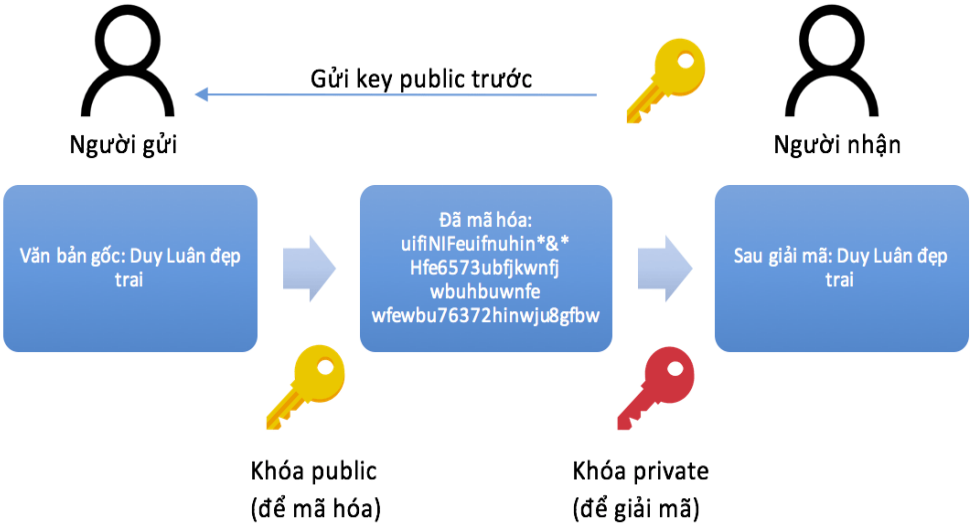
1.1.3 Các thành phần của hệ mật mã

Một hệ mật mã là một hệ bao gồm 5 thành phần (P, C, K, E, D) thoả mãn:

- P (Plaintext): không gian bản rõ, là tập hợp hữu hạn các bản rõ có thể.

- C (Ciphertext): không gian bản mã, là tập hợp những bản mã có thể.
- K (Key): không gian khoá, là tập hợp các khoá có thể.
- E (Encryption): không gian hàm mã hoá, là tập hợp các quy tắc mã hoá có thể.
- D (Decryption): không gian hàm giải mã, là tập hợp các quy tắc giải mã có thể.

Đối với mỗi k thuộc K có một quy tắc mã hoá e_k: P -> C thuộc E và một quy tắc giải mã tương ứng d_k: C -> P thuộc D



Hình 1.1 Quá trình mã hóa và giải mã thông tin

1.1.4 Phân loại hệ mật mã

Hệ mật mã đối xứng: hay còn gọi là mật mã khoá bí mật, là những hệ mật dùng chung một khó cả trong quá trình mã hoá và giải mã dữ liệu. Do đó khoá phải giữ bí mật tuyệt đối. Một số thuật toán nổi tiếng như: DES, Triple DES, RC4, …

Hệ mật mã bất đối xứng (mật mã khoá công khai): dùng một khoá để mã hoá sau đó dùng một khoá khác để giải mã (2 khoá khác nhau). Các khoá này tạo nên từng cặp chuyển đổi ngược nhau và không có khó nào có thể suy được từ khoá nào. Khóa dùng để mã hoá có thể công khai, nhưng khoá dùng để giải mã phải giữ bí mật. Một số thuật toán nổi tiếng như: RSA, Elgamal, …

1.1.5 Giới thiệu về hệ mã RSA

Hệ mã RSA là một hệ mật mã công khai được đặt tên theo ba nhà toán học Rivest, Shamir và Adleman - những người đã đề xuất nó vào năm 1978.

RSA dựa trên sự khó khăn của việc phân tích một số nguyên to lớn thành các thừa số nguyên tố, một vấn đề được gọi là bài toán phân tích nguyên tố. Đặc điểm nổi bật của RSA là tính chất hai chiều của nó: khóa công khai được sử dụng để mã hóa dữ liệu và khóa riêng tư được sử dụng để giải mã.

RSA là một trong những hệ mật mã công khai phổ biến nhất và được sử dụng rộng rãi trong các ứng dụng bảo mật mạng và giao tiếp trực tuyến. Tính an toàn của RSA dựa trên khả năng tính toán các thừa số nguyên tố của một số nguyên lớn, một nhiệm vụ mà hiện nay vẫn chưa có thuật toán hiệu quả để thực hiện trong thời gian ngắn.

So với Elgamal, RSA có ưu điểm trong việc triển khai và hiệu suất ở một số trường hợp, nhưng cũng có nhược điểm trong việc quản lý kích thước khóa và một số vấn đề khác liên quan đến bảo mật và hiệu suất. Lựa chọn giữa Elgamal và RSA thường phụ thuộc vào yêu cầu cụ thể của ứng dụng và môi trường triển khai.

1.2 Các kiến thức cơ Cơ sở toán học

Trước hết, chúng ta sẽ nhắc lại những khái niệm toán học cơ bản cần thiết cho việc hiểu RSA.

1.2.1 Số nguyên tố (prime)

Số nguyên tố là những số nguyên chỉ chia chẵn được cho 1 và cho chính nó mà thôi.

Ví dụ: 2, 3, 5, 7, 11, 13, 17, 23 ...

1.2.2 Khái niệm nguyên tố cùng nhau (relatively prime or coprime)

Với hai số nguyên dương a và b. Ta ký hiệu:

GCD (a, b): Ước chung lớn nhất của a và b (Greatest Common Divisor)

Để đơn giản ta ký hiệu: GCD (a, b) = (a, b)

Ví dụ: (4,6) = 2

(5,6) = 1

Hai số a và b gọi là nguyên tố cùng nhau khi (a, b) = 1

Ví dụ: 9 và 10 nguyên tố cùng nhau vì (9,10) = 1

1.2.3 Khái niệm modulo

Với m là một số nguyên dương. Ta nói hai số nguyên a va b là đồng dư với nhau

modulo m nếu m chia hết hiệu a-b (Viết là m|(a-b))

Ký hiệu a = b (mod m)

Như vậy a = b (mod m) khi và chỉ khi tồn tại số nguyên k sao cho a = b + km

Ví dụ:

13=3(mod 10) vì 13=3+ 1*10

1.2.4 Phi - Hàm EULER

Định nghĩa: Phi - Hàm Euler Đ(n) có giá trị tại n bằng số các số không vượt quá n và nguyên tố cùng nhau với n

Ví dụ:

Φ (5) = 4, Φ (6) = 2, Φ (10) = 4

1.2.5 Một số định lý cơ bản

Định lý Euler: nếu m là số nguyên dương và P nguyên tố cùng nhau với m thì:

P^®(m) = 1 (mod m)

Vậy nếu m và p nguyên tố cùng nhau. Ta đặt s = Đ(m) thì:

P^s = 1 (mod m)

Suy ra với a= 1 + k*s

Ta có:

P^a=P*(P^s) ^k=P*1^k(modm) = P (mod m)

với e là số nguyên dương nguyên tố cùng nhau với s, tức là (e, s) = 1

Khi đó tồn tại một nghịch đảo d của e modulo s

tức là e*d=1 (mod s) e*d = 1 + k*s

Đặt E(P) = C = P^e (mod m)

Đặt D(C) = C^d (mod m)

Ta thấy D(C) = C^d= P^e*d=P^(1+k*s) = P (mod m)

Ví dụ:

m = 10, P =9 ta có (10,9) = 1

s=¢ (10) = 4

e = 7 ta có (7,4) = 1

nghịch đảo của 7 modulo 4 la d = 3 vì 7*3 =1 + 5*4

Lúc đó ta có

E(P)=C=P^e=9^7=4.782.969=9 (mod 10) => C=9

D(C)= C^d= 9^3 = 729 = 9(mod 10)

Vậy D chính la hàm ngược của E

đây là cơ sở cho việc xây dựng thuật toán RSA mà chúng ta sẽ bàn kỹ ở phần sau.

1.3 Nội dung nghiên cứu (trình bày lý do và các nội dung nghiên cứu…)

1.3.1 Lý do nghiên cứu

Bảo mật thông tin ngày càng quan trọng trong bối cảnh các mối đe dọa mạng gia tăng. Hệ mật RSA, với cơ sở toán học vững chắc và ứng dụng rộng rãi, là một trong những giải pháp mã hóa công khai tiêu biểu. Việc nghiên cứu RSA giúp hiểu rõ cơ chế bảo mật và đóng góp vào các giải pháp an toàn thông tin trong thực tiễn.

1.3.2 Nội dung nghiên cứu

- Tổng quan về an toàn bảo mật thông tin, các mối đe dọa và giải pháp.
- Cơ sở lý thuyết và thuật toán RSA (tạo khóa, mã hóa, giải mã).
- Ưu nhược điểm của hệ mật RSA.
- Ứng dụng thực tế của RSA trong thương mại điện tử, chữ ký số và bảo mật dữ liệu.

- Đề xuất cải tiến và thảo luận triển vọng nghiên cứu.

CHƯƠNG 2 KẾT QUẢ NGHIÊN CỨU

2.1 Nghiên cứu, tìm hiểu hệ mã hóa khóa công khai

2.1.1 Giới thiệu chung về hệ mã RSA

- Hệ mã RSA là một hệ mật mã công khai được đặt tên theo ba nhà toán học Rivest, Shamir và Adleman - những người đã đề xuất nó vào năm 1978.

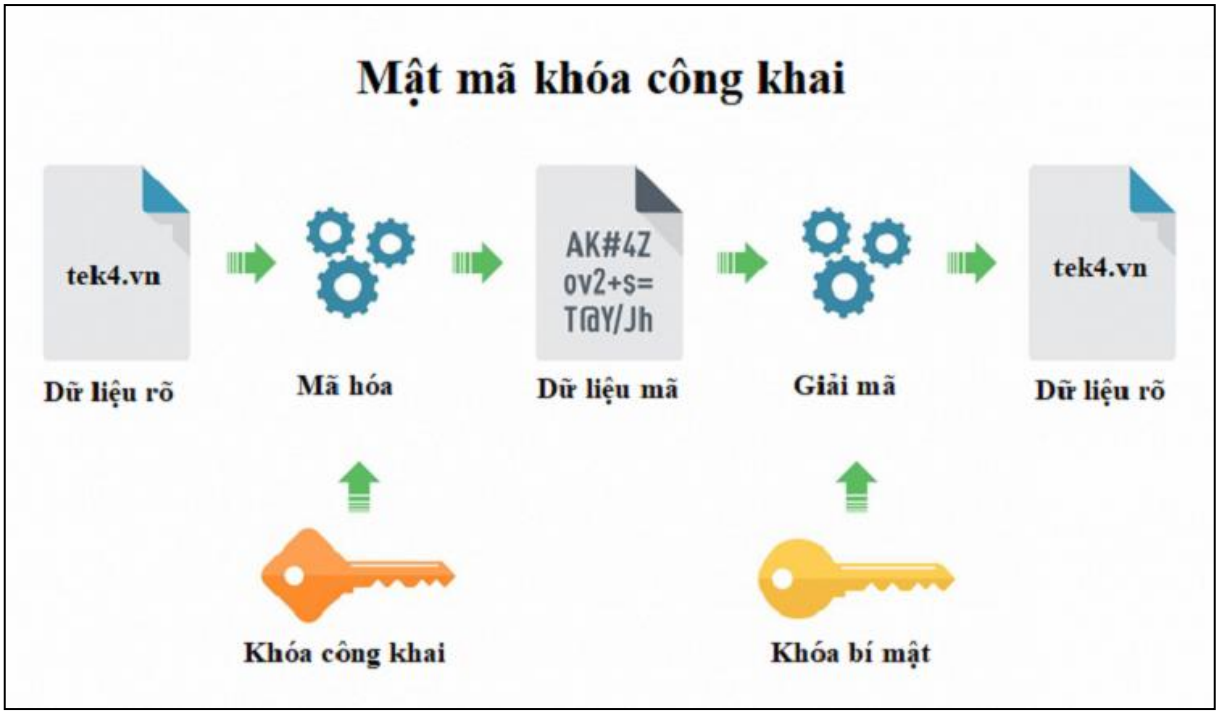
- RSA dựa trên sự khó khăn của việc phân tích một số nguyên to lớn thành các thừa số nguyên tố, một vấn đề được gọi là bài toán phân tích nguyên tố. Đặc điểm nổi bật của RSA là tính chất hai chiều của nó: khóa công khai được sử dụng để mã hóa dữ liệu và khóa riêng tư được sử dụng để giải mã.

- RSA là một trong những hệ mật mã công khai phổ biến nhất và được sử dụng rộng rãi trong các ứng dụng bảo mật mạng và giao tiếp trực tuyến. Tính an toàn của RSA dựa trên khả năng tính toán các thừa số nguyên tố của một số nguyên lớn, một nhiệm vụ mà hiện nay vẫn chưa có thuật toán hiệu quả để thực hiện trong thời gian ngắn.

- So với Elgamal, RSA có ưu điểm trong việc triển khai và hiệu suất ở một số trường hợp, nhưng cũng có nhược điểm trong việc quản lý kích thước khóa và một số vấn đề khác liên quan đến bảo mật và hiệu suất. Lựa chọn giữa Elgamal và RSA thường phụ thuộc vào yêu cầu cụ thể của ứng dụng và môi trường triển khai.

2.1.2 Mã hoá và giải mã hệ mã RSA

Quá trình mã hóa và giải mã



Hình 2.1 Quá trình mã hóa và giải mã thông tin

Quá trình mã hóa và giải mã trong hệ mã RSA diễn ra như sau:

Tạo khóa:

- Chọn các tham số cơ bản:
  - Chọn hai số nguyên tố lớn p và q.
  - Tính n = p × q, $\phi(n) = (p-1) \times (q-1)$ làm modulus của hệ thống.
  - Chọn một số nguyên e thỏa mãn điều kiện 1 <e <$\phi(n)$, trong đó e là nguyên tố cùng nhau với $\phi(n)$. e là phần tử công khai.
  - Tính khoá bí mật d sao cho $d \times e \equiv 1 \mod \phi(n)$.
- Tạo khoá công khai và khoá bí mật:
  - Khoá công khai: Kpub = {e, n}
  - Khoá bí mật: Kpr = {d, n}

Mã hóa:

- Mã hoá thông điệp:
    - Chuyển thông điệp M thành một số nguyên m sao cho $0 \leq m < n$.
    - Tính bản mã $C = m^e \bmod n$.

Giải mã:

- Sử dụng khoá bí mật d:
- Tính thông điệp M: Sử dụng d và công thức: $M = C^d \bmod n$.

2.1.3 Độ an toàn

Độ an toàn của hệ thống RSA dựa trên hai vấn đề: bài toán phân tích ra thừa số nguyên tố các số nguyên lớn và bài toán RSA.

- Vì vậy, muốn xây dựng hệ RSA an toàn thì n=p*q phải là một số đủ lớn, để không có khả năng phân tích nó về mặt tính toán. Để đảm bảo an toàn nên chọn các số nguyên tố p và q từ 100 chữ số trở lên.

Bảng 2.1 Thời gian phân tích thừa số nguyên tố

Số các chữ số trong số được phân tích	Thời gian phân tích
50	4 giờ
75	104 giờ
100	74 năm
200	4000 năm

300	500000 năm
500	4 x 10 ^ 25 năm

Bảng 1.1. Bảng thời gian phân tích mã RSA

- Cách thức phân phối khóa công khai là một trong những yếu tố quyết định đối với độ an toàn của RSA.

- Vấn đề này nảy sinh ra một lỗ hổng gọi là Man-in-the-middle attack (tấn công vào giữa).

  - Khi A và B trao đổi thông tin thì C có thể gửi cho A một khóa bất kì để A tin rằng đó là khóa công khai của B gửi.
  - Sau đó C sẽ giải mã và đánh cắp được thông tin. Đồng thời mã hóa lại thông tin theo khóa công khai của B và gửi lại cho B.
  - Về nguyên tắc, cả A và B đều không phát hiện được sự can thiệp của C.

2.2 Nghiên cứu, tìm hiểu về hệ mật mã RSA

2.2.1 Nội dung thuật toán

2.2.1.1 Thuật toán Euclid tìm ước chung lớn nhất

- Thuật toán Euclid tìm ước chung lớn nhất của hai số nguyên không âm GCD(a, b)

Nếu a = 0 thì GCD (a, b) = b return b

Nếu b = 0 thì GCD (a, b) = a return a

Nếu a ≠ 0, b ≠ 0. Giả sử A > B:

Viết a dưới dạng: a = q * b + r

Theo thuật toán Euclid ta có:

GCD (a, b) = GCD (b, r) = GCD(b, a mod b)

```
// Hàm tính ước số chung lớn nhất (GCD) của hai số a và b
int gcd(int a, int b) {
    if (b == 0) // Nếu b = 0, thì GCD là a
        return a;
    return gcd(b, a % b); // Đệ quy tính GCD theo thuật toán Euclid
}
```

Hình 2.2 Demo thuật toán Euclid

2.2.1.2 Thuật toán Euclid mở rộng tìm phần tử nghịch đảo

Cho r0 ,r1 tồn tại 2 số nguyên s,t sao cho s.r0+t.r1=GCD(r0 ,r1)=d

Nếu d>1 thì r1-1 mod r0 không tồn tại

Nếu d=1 return t

Công thức tính s và t:

s0 = 1 t0 = 0

s1 = 0 t1 = 1

si = si-2 – qi-1*si-1 ti = ti-2 – qi-1*ti-1

Trong đó ta có : với i = 0,1,2,3…

ri = ri+2 + qi+1*ri+1

Dừng lại khi ri+2 =0

```
// Hàm tìm nghịch đảo modulo của a theo m
int find_A(int a, int m) {
    int m0 = m, y = 0, x = 1; // Khởi tạo các giá trị cần thiết

    if (m == 1) return 0; // Nếu m = 1, nghịch đảo là 0

    // Sử dụng thuật toán Euclid mở rộng để tìm nghịch đảo modulo
    while (a > 1) {
        int q = a / m; // Chia a cho m
        int t = m;     // Lưu trữ giá trị của m

        m = a % m;     // Cập nhật m là phần dư của a chia cho m
        a = t;         // Cập nhật a là giá trị cũ của m
        t = y;

        y = x - q * y; // Cập nhật giá trị y
        x = t;         // Cập nhật giá trị x
    }

    // Nếu x là âm, thì thêm m vào để đảm bảo x dương
    if (x < 0)
        x += m0;

    return x; // Trả về nghịch đảo modulo
}
```

Hình 2.3 Demo thuật toán Euclid mở rộng

2.2.1.3 Định lí Ferma

- Định lý Fermat được phát biểu như sau: Nếu p là số nguyên tố và a là số nguyên dương không thể chia hết cho p, thì: ap-1 ≡1(mod p)

Ví dụ: a = 5; p =3

Ta có ap-1 = 53-1=52 = 25≡1(mod p) = ≡1(mod 3)

Chứng minh: Ta xét các số nhỏ hơn p: {1, 2, ..., p-1} và nhân mỗi phần tử với a, (modulo p) để có được tập X = {a mod p, 2a mod p, ..., (p-1)a mod p }. Không phần tử nào của X bằng 0 bởi vì a không chia hết cho p. Hơn nữa không

có hai số nguyên nào của X bằng nhau. Để xét điều này ta giả sử: ja ≡ ka (mod p), với 1≤j

a × 2a ×....× (p-1)a ≡ [(1×2×.....×(p-1)](mod p)

a p-1 (p-1)! ≡(p-1)!(mod p) Bỏ (p-1)! => ap-1 ≡ 1(mod p)

Một dạng thay thế của định lý Fermat: Nếu p là số nguyên tố, a là một số nguyên dương thì:

a p ≡ a (mod p)

Lưu ý rằng hình thức đầu của định lý Fermat: ap-1 ≡ 1(mod p) yêu cầu rằng a tương đối nguyên tố với p, nhưng dạng này không đúng

p=5; a = 3; ap = 35 = 243 ≡ 3(mod 5) = a(mod p) p=5; a=10; ap

= 105 = 100000 ≡ 10 (mod 5) ≡ 0(mod 5) = a(mod 5)

2.2.1.4 Định lí Eule

Cho số nguyên dương n, số lượng các số nguyên dương bé hơn n và nguyên tố cùng nhau với n được ký hiệu f(n) và gọi là hàm Euler.

Nhận xét: Nếu p là số nguyên tố, thì f(p) = p -1

Ví dụ: Tập các số nguyên không âm nhỏ hơn 7 là Z7 = {0, 1, 2, 3, 4, 5, 6}.

Do 7 là số nguyên tố, nên tập các số nguyên dương nhỏ hơn 7 và nguyên tố cùng nhau với 7 là Z7* = {1, 2, 3, 4, 5, 6}.

Khi đó |Z| = f(p) = p - 1 = 7 - 1 = 6

Định lý về hàm Euler:

Nếu n là tích của hai số nguyên tố p, q thì f(n) = f(p). f(q) = (p-1).(q-1)

Quan hệ Đồng dư

Khái niệm: Cho hai số nguyên a, b, m (m > 0). Ta nói rằng a và b "đồng dư" với nhau theo modulo m, nếu chia a và b cho m, ta nhận được cùng một số dư.

Ký hiệu: a º b (mod m).

Ví dụ: 17 º 5 (mod 3) vì chia 17 và 5 cho 3, được cùng số dư là 2.

Tính chất của quan hệ đồng dư:

- Với mọi số nguyên dương m ta có:
  - a ≡ a (mod m) với mọi a thuộc Z; (tính chất phản xạ)
  - a ≡ b (mod m) thì b ≡ a (mod m); (tính chất đối xứng)
  - a≡ b (mod m) và b≡c (mod m) thì a≡c (mod m); (tính chất bắc cầu)
- Tổng hay hiệu các đồng dư:
  - (a+b) (mod n) ≡ [(a mod n) + (b mod n)] (mod n)
  - (a-b) (mod n) ≡ [(a mod n) - (b mod n)] (mod n)
- Tích các đồng dư:
  - (a*b) (mod n) ≡ [(a mod n) * (b mod n)] (mod n)

Tập thặng dư thu gọn theo modulo

Khái niệm: Kí hiệu Z n = {0, 1, 2, ..., n-1} là tập các số nguyên không âm < n.

Với Z n và phép cộng (+) lập thành nhóm Cyclic có phần tử sinh là 1, phần tử trung lập e = 0, (Z n, +) gọi là nhóm cộng, đó là nhóm hữu hạn có cấp n.

Với phép (+) là phép cộng thông thường của các số nguyên.

Kí hiệu Z*n = {x Î Z n, x là nguyên tố cùng nhau với n}. Tức là x phải ¹ 0.

Z*n được gọi là Tập thặng dư thu gọn theo mod n, có số phần tử là f(n). Z*n với phép nhân mod n lập thành một nhóm (nhóm nhân), phần tử trung lập e = 1. Tổng quát (Z*n, phép nhân mod n) không phải là nhóm Cyclic. Nhóm nhân Z*n là Cyclic chỉ khi n có dạng: 2, 4, p k hay 2p k với p là nguyên tố lẻ.

Ví dụ: Cho n = 21, Z*n = {1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20}.

Độ phức tạp của thuật toán

Bài toán được diễn đạt bằng hai phần:

Đầu vào: Các dữ liệu vào của bài toán.

Đầu ra: Các dữ liệu ra của bài toán (kết quả).

Khái niệm Thuật toán: "Thuật toán" được hiểu đơn giản là cách thức để giải một bài toán. Cũng có thể được hiểu bằng hai quan niệm: trực giác hay hình thức như sau:

Một cách trực giác, Thuật toán được hiểu là một dãy hữu hạn các quy tắc (chỉ thị, mệnh lệnh) mô tả một quá trình tính toán, để từ dữ liệu đã cho đầu vào ta nhận được kết quả đầu ra của bài toán.

Một cách hình thức, người ta quan niệm thuật toán là một máy Turing. Thuật toán được chia thành hai loại: Đơn định và không đơn định. Chi phí thời gian của một quá trình tính toán là thời gian cần thiết để thực hiện một quá trình tính toán. Với thuật toán tựa Algol: Chi phí thời gian là số các phép tính cơ bản thực hiện trong quá trình tính toán.

Chi phí bộ nhớ của một quá trình tính toán là số ô nhớ cần thiết để thực hiện một quá trình tính toán. Gọi A là thuật toán, e là dữ liệu vào của bài toán đã được mã hóa bằng cách nào đó. Thuật toán A tính trên dữ liệu vào e phải trả một giá nhất định. Ta ký hiệu: tA(e) là giá thời gian và IA(e) là giá bộ nhớ.

2.2.1.5 Thuật toán Miller - Rabin

Miller-Rabin là một thuật toán xác suất để kiểm tra tính nguyên tố cũng như các thuật toán kiểm tra tính nguyên tố: Kiểm tra Fermat và Kiểm tra Solovay-Strassen. Nó được đề xuất đầu tiên bởi Gary L. Miller như một thuật toán tất định, dựa trên giả thiết Riemann tổng quát; Michael O. Rabin đã sửa chữa nó thành một thuật toán xác suất.

Giải thuật:

Giả sử ta có số:

n= p*q. Với p, q là các số nguyên tố.

Với một bài toán cho biết số n rồi tìm 2 số p và q, thường hay là nhất là bạn sẽ chia n cho các số từ 2 đến √n, từ đó bạn có thể xác định được hai số p và q. Nhưng khi số n là một số cực lớn thì việc này trở nên khó khăn rất nhiều và khi chạy trong một thời gian quá dài. Thay vào đó Miller đã lựa chọn một số nguyên x bất kì thỏa mãn 1<x<n, và UCLN(x,n) =1. Số nguyên x đó ta gọi là seed (hạt). Tìm số nguyên dương w nhỏ nhất và thỏa mãn:

$X^w \bmod n = 1$

Gọi w là order of the seed x (bậc của hạt x). Nếu chỉ ra rằng w là một số lẻ tương ứng với x, khi đó sẽ thử lại với một số x khác, cho đến khi x là một số chẵn. Tiếp theo, tính:

$M = X^{w/2}$

Và với M là một số nguyên. Sau đó, tính tích của A = M-1 và B= M+1. Nếu B mod n = 0, tìm x thỏa mãn hai điều kiện:

(1) w là số chẵn

(2) B mod n khác 0

Cuối cùng, để xác định được ước chung lớn nhất của A, n và B và n. Có:

p = UCLN(A,n) và q = UCLN(B,n).

Ví dụ:

Tìm tích của hai số nguyên tố p và q với n = p.q = 15. Với các số: $x_1 = 1$, $x_2 = 2$, $x_3 = 4$, $x_4 = 7$, $x_5 = 8$, $x_6 = 11$, $x_7 = 13$, và $x_8 = 14$

Dễ thấy UCLN(x_i,15) =1 với I thuộc (1,8) Chọn x = x_2 = 2 Ta sẽ có: 2*2 mod 15 = 4 mod 15 =4. 4*2 mod 15 = 8 mod 15 =8. 8*2 mod 15 = 16 mod 15 =1. Do đó, ta thấy 2^4 mod 15=1 suy ra w =4, cũng là số mũ nhỏ nhất thỏa mãn:

- 2^w mod 15 = 1
- A = $2^{w/2}$ - 1 = 3 và B = 2w/2 + 1 = 5.

Dễ thấy w =4 thỏa mãn 2 điều kiện w là số chẵn và B mod n khác 0. Và ta có p = UCLN(A,15) = UCLN(3,15) =3 và

q = UCLN(B,15)= UCLN(5,15) =5.

Do đó 15 phân tích ra thừa số nguyên tố là 3 và 5.

2.2.2 Phương pháp tấn công

Phân phối khóa:

Cũng giống như các thuật toán mã hóa khác, cách thức phân phối khóa công khai là một trong những yếu tố quyết định đối với độ an toàn của RSA. Quá trình phân phối khóa cần chống lại được tấn công đứng giữa (man-in-the-middle attack). Giả sử kẻ xấu(C) có thể gửi cho Người gửi thông tin(A) một khóa bất kỳ và khiến (A) tin rằng đó là khóa (công khai) của Đối tác(B). Đồng thời (C) có khả năng đọc

được thông tin trao đổi giữa (A) và (B). Khi đó, (C) sẽ gửi cho (A) khóa công khai của chính mình (mà (A) nghĩ rằng đó là khóa của (B)). Sau đó, (C) đọc tất cả văn

bản mã hóa do (A) gửi, giải mã với khóa bí mật của mình, giữ 1 bản copy đồng thời mã hóa bằng khóa công khai của (B) và gửi cho (B). Về nguyên tắc, cả (A) và (B) đều không phát hiện ra sự can thiệp của người thứ ba.

Các phương pháp chống lại dạng tấn công này thường dựa trên các chứng thực khóa công khai (digital certificate) hoặc các thành phần của hạ tầng khóa công khai (public key infrastructure - PKI).

Tấn công dưa trên thời gian:

Vào năm 1995, Paul Kocher mô tả một dạng tấn công mới lên RSA: nếu kẻ tấn công nắm đủ thông tin về phần cứng thực hiện mã hóa và xác định được thời gian giải mã đối với một số bản mã lựa chọn thì có thể nhanh chóng tìm ra khóa d. Dạng tấn công này có thể áp dụng đối với hệ thống chữ ký điện tử sử dụng RSA. Năm 2003, Dan Boneh và David Brumley chứng minh một dạng tấn công thực tế hơn: phân tích thừa số RSA dùng mạng máy tính (Máy chủ web dùng SSL). Tấn công đã khai thác thông tin rò rỉ của việc tối ưu hóa định lý số dư Trung quốc mà nhiều ứng dụng đã thực hiện.

Để chống lại tấn công dựa trên thời gian là đảm bảo quá trình giải mã luôn diễn ra trong thời gian không đổi bất kể văn bản mã. Tuy nhiên, cách này có thể làm giảm hiệu suất tính toán. Thay vào đó, hầu hết các ứng dụng RSA sử dụng một kỹ thuật gọi là che mắt. Kỹ thuật này dựa trên tính nhân của RSA: thay vì tính cd mod n, Alice đầu tiên chọn một số ngâu nhiên r và tính (rec)d mod n. Kết quả của phép tính này là rm mod n và tác động của r sẽ được loại bỏ bằng cách nhân kết quả với nghịch đảo của r. Đỗi với mỗi văn bản mã, người ta chọn một giá trị của r. Vì vậy, thời gian giải mã sẽ không còn phụ thuộc vào giá trị của văn bản mã.

Tấn công bằng phương pháp lựa chọn thích nghi bản mã:

Năm 1981, Daniel Bleichenbacher mô tả dạng tấn công lựa chọn thích nghi bản mã (adaptive chosen ciphertext attack) đầu tiên có thể thực hiện trên thực tế đối với một văn bản mã hóa bằng RSA. Văn bản này được mã hóa dựa trên tiêu chuẩn PKCS #1 v1, một tiêu chuẩn chuyển đổi bản rõ có khả năng kiểm tra tính hợp lệ của văn bản sau khi giải mã. Do những khiếm khuyết của PKCS #1, Bleichenbacher có thể thực hiện một tấn công lên bản RSA dùng cho giao thức SSL (tìm được khóa phiên). Do phát hiện này, các mô hình chuyển đổi an toàn hơn như chuyển đổi mã hóa bất đối xứng tối ưu (Optimal Asymmetric Encryption Padding) được khuyến cáo sử dụng. Đồng thời phòng nghiên cứu của RSA cũng đưa ra phiên bản mới của PKCS #1 có khả năng chống lại dạng tấn công nói trên.

2.2.3 Đánh giá độ phức tạp của thuật toán

Sinh khóa RSA:

- Chọn hai số nguyên tố lớn p và q: Đây là bước đầu tiên và quan trọng nhất của quá trình sinh khóa. Độ phức tạp của việc này phụ thuộc vào thuật toán được sử dụng để tìm số nguyên tố lớn và kiểm tra tính nguyên tố của chúng. Các thuật toán phổ biến như tìm kiếm nguyên tố ngẫu nhiên và sàng Eratosthenes có độ phức tạp trung bình là O(sqrt(n)/log(n)), trong đó n là số bit của số nguyên tố cần tìm.

- Tính toán n = p * q: Phép nhân hai số nguyên tố lớn có độ phức tạp gần như là O(n^2), trong đó n là số bit của p và q.

- Chọn số e và d: Đối với khóa công khai e, việc chọn một số nguyên tố tương đối nhỏ và tìm số nguyên dư khi chia cho (p-1) *(q-1) có thể được thực hiện trong thời gian O(log(n)). Tuy nhiên, đối với khóa bí mật d, việc

tính toán d từ e, p và q có thể mất thời gian nhiều hơn, có thể lên đến O(n^3) nếu sử dụng thuật toán mở rộng Euclid.

Mã hóa và giải mã RSA:

- **Mã hóa:** Phép tính mã hóa RSA là một phép lũy thừa modul exponentiation, được thực hiện thông qua thuật toán Square and Multiply hoặc Montgomery Exponentiation. Độ phức tạp của phép tính này là O(log(n)^3) hoặc O(log(n)^4) tùy thuộc vào phương pháp thực hiện.

- **Giải mã:** Phép tính giải mã cũng là một phép lũy thừa modul exponentiation, nhưng với khóa bí mật d. Độ phức tạp của phép tính này cũng là O(log(n)^3) hoặc O(log(n)^4).

2.2.4 Ưu điểm và nhược điểm của hệ mã RSA

Ưu điểm của hệ mã RSA:

- An toàn và bảo mật cao.
- Đơn giản và dễ triển khai.
- Linh hoạt trong việc mã hóa và tạo chữ ký điện tử.

Nhược điểm của hệ mã RSA:

- Tính chậm trong quá trình xử lý số nguyên lớn.
- Yêu cầu dung lượng khóa lớn.
- Định dạng khóa phức tạp và việc quản lý khóa đòi hỏi kiến thức chuyên sâu.

2.2.5 Hướng phát triển

Đứng trước xu hướng phát triển mạnh mẽ của Công nghệ thông tin việc nghiên cứu các vấn đề bảo mật và an toàn thông tin để đảm bảo cho các hệ thống

thông tin hoạt động an toàn là điều vô cùng quan trọng và bức thiết. Với đề tài xây dựng chương trình mã hóa và giải mã RSA, tuy chương trình cài đặt chưa được hoàn hảo, song nếu có thời gian phát triển và hoàn thiện hơn thì chương trình sẽ rất có ích và có tính ứng dụng cao.

- Chứng thực (authentication) là quá trình một server xác định xem bạn là ai. Phương pháp chứng thực đơn giản nhất mà bạn sử dụng hằng ngày được gọi là "đăng nhập" (log in). Hằng ngày, bạn nhập một mật khẩu tĩnh (không thay đổi trong vài tháng, hoặc cả đời), và server sẽ dò xem mật khẩu của bạn có khớp với mật khẩu được lưu hay không. Tuy nhiên, một số người đặt mật khẩu rất dễ đoán, và mật khẩu có khả năng bị đánh cắp rất cao bằng những thủ thuật đơn giản. Do đó, trong một số các ứng dụng, một "mật khẩu điện tử" sẽ giúp việc chứng thực trở nên an toàn hơn. Phương thức chứng thực này thường được sử dụng trong SSH để đăng nhập vào máy khác (thường là server) mà không cần phải nhập mật khẩu.

- Ghi chú: Server không lưu trực tiếp mật khẩu của bạn, mà lưu bản hash của nó, để nếu khi hash bị lộ thì mật khẩu của bạn vẫn tương đối an toàn.

- Thay vì một mật khẩu bình thường, người đăng nhập sẽ sinh ra một ra một bộ khóa công khai và bí mật. Sau đó, người đăng nhập sẽ đăng ký khóa công khai lên server (khóa bí mật không bao giờ bị tiết lộ ra ngoài). Mỗi khi cần đăng nhập vào server, các bước sau đây sẽ xảy ra:
    - Server sinh ra một số ngẫu nhiên a và gửi cho người đăng nhập.
    - Người đăng nhập sử dụng khóa bí mật để tạo mật văn b gửi lại cho server.

- Server sử dụng khóa công khai (được đăng ký trước đó) để giải mã. Nếu kết quả sau khi giải mã lại bằng a, đó là bằng chứng cho việc người đăng nhập sở hữu đúng khóa bí mật.

- Nếu một kẻ mạo nhận muốn đăng nhập vào server, sau khi được nhận a từ server, hắn không có đúng khóa bí mật và gửi lại mật văn giả b', thì khi server giải mã ra lại sẽ có a' khác a và do đó hắn không thể đăng nhập. Mặt khác, hắn không thể đoán hay tái sử dụng mật văn b, bởi vì a là ngẫu nhiên và sẽ thay đổi cho mỗi lần đăng nhập khác nhau.

- Một điều bất tiện khi sử dụng "mật khẩu điện tử" này là khóa bí mật chỉ hợp lệ trên một máy duy nhất (bạn có thể copy sang máy khác, nhưng điều đó làm khóa bí mật của bạn gặp nhiều rủi ro bị tiết lộ hơn). Tuy nhiên, bạn hoàn toàn có thể tạo một bộ khóa riêng cho mỗi máy và đăng ký nhiều khóa công khai cho một tài khoản, như vậy bạn có thể đăng nhập ở bất kỳ máy nào.

- Tuy nhiên, nếu kẻ tấn công nghe lén những gói tin của bạn, mặc dù hắn không biết được mật khẩu, hắn vẫn có thể biết được những thông tin bạn truyền sau khi bạn đăng nhập, ví dụ lịch sử giao dịch ngân hàng của bạn. Do đó, bạn cần phải có một cách để bảo vệ đường truyền của bạn.

2.3 Thiết kế chương trình, cài đặt thuật toán.

2.3.1 Thiết kế kịch bản chương trình

- Ở giao diện sinh khóa, người dùng kích vào nút tạo khóa, chương trình sẽ sinh tự động khóa bí mật và khóa công khai có kích thước lớn.

- Người dùng nhập chuỗi văn bản vào ô input bản rõ hoặc có thể mở đọc file .txt hoặc .docx để hiển thị nội dung lên màn hình

- Người dùng kích vào nút “Mã hóa”, chương trình sẽ mã hóa chuỗi bản rõ và hiển thị chuỗi sau khi mã hóa và ô input bản mã. Người dùng có thể kích vào nút “Lưu” để lưu bản mã

- Người dùng kích và nút “Giải mã”, chương trình tiến hành giải mã và hiển thị nội dung văn bản sau khi giải mã lên mà hình. Nội dung sau khi giải mã giống với nội dung bản rõ ban đầu, chương trình giải mã thành công. Người dùng kích nút “Lưu” để lưu văn bản được giải vào file.

2.3.2 Giới thiệu ngôn ngữ lập trình sử dụng để cài đặt thuật toán.

Ngôn ngữ lập trình được sử dụng để cài đặt thuật toán là C++ và C#:

- **Ngôn ngữ C++**: là một lựa chọn mạnh mẽ và phổ biến để cài đặt thuật toán RSA nhờ hiệu suất cao và khả năng quản lý tài nguyên tốt. Với các thư viện tích hợp sẵn như <iostream> và <cmath>, cùng các cấu trúc điều khiển linh hoạt, C++ cho phép chúng ta triển khai thuật toán một cách dễ dàng và hiệu quả. Bên cạnh đó QT creation sẽ được sử dụng để thiết kế giao diện cho chương trình cài đặt hệ mã RSA.

Các ưu điểm của C++ khi cài đặt RSA bao gồm:

  - Hiệu suất cao: Thích hợp để xử lý các phép toán số học lớn.
  - Khả năng quản lý bộ nhớ: Cho phép kiểm soát chi tiết các hoạt động cấp phát bộ nhớ, tối ưu hóa hiệu năng.
  - Thư viện tiêu chuẩn mạnh mẽ: Hỗ trợ các phép toán cơ bản và nâng cao như số mũ, modulo, kiểm tra số nguyên tố.

- **Ngôn ngữ C#:** là một lựa chọn hiện đại và dễ sử dụng để triển khai thuật toán RSA, đặc biệt trong các ứng dụng hướng đối tượng. Với sự hỗ

trợ từ .NET Framework, C# cung cấp nhiều lớp và phương thức sẵn có, giúp việc thực hiện các phép toán phức tạp như số mũ và modulo trở nên đơn giản hơn.

Các ưu điểm của C# khi cài đặt RSA bao gồm:

- Thao tác số học đơn giản: Hỗ trợ các kiểu dữ liệu như BigInteger từ thư viện System.Numerics, giúp xử lý các số lớn cần thiết trong RSA.
- Hướng đối tượng mạnh mẽ: Dễ dàng tổ chức mã nguồn và tái sử dụng mã.
- Hỗ trợ nền tảng .NET: Tích hợp các công cụ và thư viện để xây dựng giao diện người dùng hoặc tích hợp thuật toán vào các ứng dụng lớn hơn.

2.3.3 Cài đặt thuật toán, giao diện chương trình

2.3.3.1 Cài đặt chương trình bằng c++

a. Thuật toán

```
// Hàm tính ước số chung lớn nhất (GCD) của hai số a và b
int gcd(int a, int b) {
    if (b == 0) // Nếu b = 0, thì GCD là a
        return a;
    return gcd(b, a % b); // Đệ quy tính GCD theo thuật toán Euclid
}
```

Hình 2.4 Hàm tính ước chung lớn nhất GCD()

```
// Hàm tìm nghịch đảo modulo của a theo m
int find_A(int a, int m) {
    int m0 = m, y = 0, x = 1; // Khởi tạo các giá trị cần thiết

    if (m == 1) return 0; // Nếu m = 1, nghịch đảo là 0

    // Sử dụng thuật toán Euclid mở rộng để tìm nghịch đảo modulo
    while (a > 1) {
        int q = a / m; // Chia a cho m
        int t = m;     // Lưu trữ giá trị của m

        m = a % m;     // Cập nhật m là phần dư của a chia cho m
        a = t;         // Cập nhật a là giá trị cũ của m
        t = y;

        y = x - q * y; // Cập nhật giá trị y
        x = t;         // Cập nhật giá trị x
    }

    // Nếu x là âm, thì thêm m vào để đảm bảo x dương
    if (x < 0)
        x += m0;

    return x; // Trả về nghịch đảo modulo
}
```

Hình 2.5 Hàm tìm nghịch đảo modulo find_A()

```
// Hàm kiểm tra số nguyên tố
bool isPrime(int n) {
    if (n < 2) return false; // Số nhỏ hơn 2 không phải là số nguyên tố
    for (int i = 2; i * i <= n; i++) { // Kiểm tra chia hết cho các số từ 2 đến căn bậc 2 của n
        if (n % i == 0) return false; // Nếu n chia hết cho i, thì n không phải là số nguyên tố
    }
    return true; // Nếu không tìm thấy số nào chia hết, n là số nguyên tố
}
```

Hình 2.6 Hàm kiểm tra số nguyên tố isPrime()

```
// Hàm sinh số nguyên tố ngẫu nhiên trong khoảng từ 50 đến 150
int generatePrime() {
    int candidate = rand() % 5000 + 1000; // Sinh số ngẫu nhiên trong khoảng [50, 150]
    while (!isPrime(candidate)) { // Kiểm tra xem số này có phải là số nguyên tố không
        candidate = rand() % 5000 + 1000; // Nếu không phải, tiếp tục sinh số ngẫu nhiên khác
    }
    return candidate; // Trả về số nguyên tố đã tìm được
}
```

Hình 2.7 Hàm sinh số nguyên tố lớn generatePrime()

```
// Hàm tính lũy thừa theo mô-đun sử dụng đệ quy
long long power_recursive(long long a, int n, int m) {
    if (n == 0) return 1; // Nếu n = 0, lũy thừa của a là 1
    long long half = power_recursive(a, n / 2, m); // Tính lũy thừa của a^(n/2) mod m
    half = (half * half) % m; // Tính (a^(n/2))^2 mod m

    if (n % 2 != 0) { // Nếu n là số lẻ
        half = (half * a) % m; // Nhân thêm a để hoàn thành lũy thừa
    }
    return half; // Trả về kết quả lũy thừa theo mô-đun
}
```

Hình 2.8 Hàm tính lũy thừa modulo

```
// Hàm sinh khóa công khai và khóa riêng trong hệ thống RSA
void sinkhoa(int &numberN, int &numberB, int &numberA) {
    int primeA = generatePrime(); // Sinh số nguyên tố P
    int primeB = generatePrime(); // Sinh số nguyên tố Q
while (primeA == primeB) { // Đảm bảo rằng P và Q khác nhau
        primeB = generatePrime(); // Nếu P = Q, tiếp tục sinh số mới cho Q
    }

    numberN = primeA * primeB; // Tính N = P * Q
    int numberPhiN = (primeA - 1) * (primeB - 1); // Tính Phi(N) = (P - 1) * (Q - 1)

    numberB = 3; // Chọn giá trị khóa công khai B ban đầu là 3
    while (gcd(numberB, numberPhiN) != 1) { // Tìm B sao cho GCD(B, Phi(N)) = 1
        numberB++; // Tăng giá trị B cho đến khi GCD(B, Phi(N)) = 1
    }

    numberA = find_A(numberB, numberPhiN); // Tính khóa riêng A bằng cách tìm nghịch đảo của B mod Phi(N)

    // Hiển thị các giá trị P, Q, N, Phi(N), B (khóa công khai) và A (khóa riêng)
    cout << "P = " << primeA << ", Q = " << primeB << endl;
    cout << "N = " << numberN << ", Phi(N) = " << numberPhiN << endl;
    cout << "Public Key (B) = " << numberB << endl;
    cout << "Private Key (A) = " << numberA << endl;
}
```

Hình 2.9 Hàm sinh khóa công khai và khóa riêng

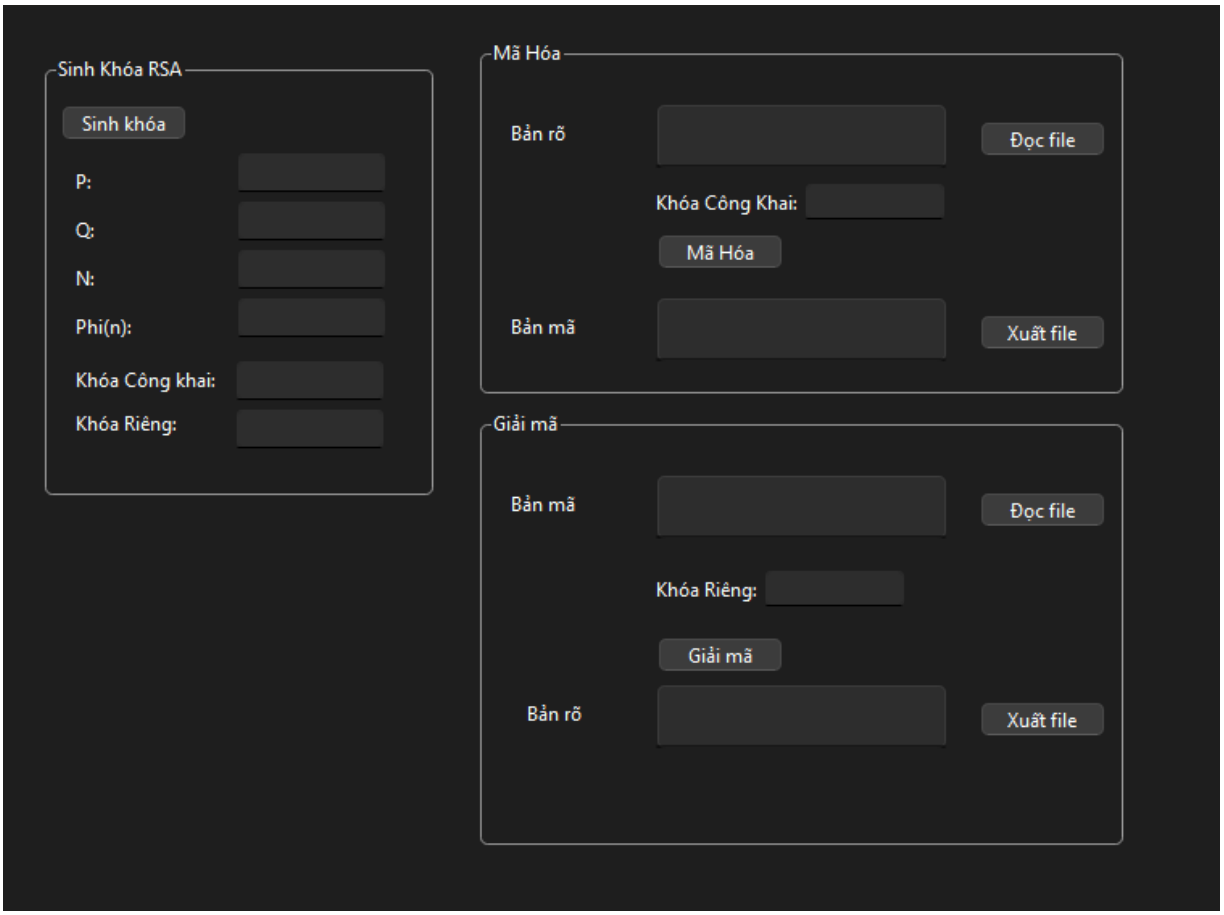
```
// Hàm mã hóa chuỗi văn bản với khóa công khai
string MaHoa(const string &strPlainText, int numberB, int numberN) {
    string encryptedMessage; // Chuỗi lưu trữ bản mã

    cout << "Encrypted Message: ";
    for (char ch : strPlainText) { // Duyệt qua từng ký tự trong văn bản gốc
        int intAscii = static_cast<int>(ch); // Chuyển ký tự thành mã ASCII
        int encryptedChar = power_recursive(intAscii, numberB, numberN); // Mã hóa ký tự theo công thức RSA
        encryptedMessage += to_string(encryptedChar) + " "; // Lưu số mã hóa vào chuỗi
        cout << encryptedChar << " "; // Hiển thị số mã hóa ra màn hình
    }
    cout << endl;

    return encryptedMessage; // Trả về chuỗi bản mã
}
```

Hình 2.10 Hàm mã hóa chuỗi văn bản

```
// Hàm giải mã chuỗi đã mã hóa với khóa riêng
string GiaiMa(const string &encryptedMessage, int numberA, int numberN) {
    string strDecryptedMessage; // Chuỗi lưu trữ thông điệp đã giải mã
    string temp;
    vector<int> encryptedNumbers; // Mảng lưu trữ các số nguyên từ bản mã

    // Tách các số nguyên từ chuỗi bản mã
    for (char ch : encryptedMessage) {
        if (ch == ' ') { // Nếu gặp khoảng trắng, tách số nguyên
            encryptedNumbers.push_back(stoi(temp)); // Chuyển chuỗi thành số và thêm vào mảng
            temp.clear(); // Làm sạch chuỗi tạm
        } else {
            temp += ch; // Nếu không phải khoảng trắng, thêm ký tự vào chuỗi tạm
        }
    }

    // Giải mã các số nguyên thành ký tự
    cout << "Decrypted Message: ";
    for (int encryptedChar : encryptedNumbers) {
        int decryptedChar = power_recursive(encryptedChar, numberA, numberN); // Giải mã số nguyên theo khóa riêng
        cout << static_cast<char>(decryptedChar); // Chuyển số nguyên đã giải mã thành ký tự và hiển thị
strDecryptedMessage += static_cast<char>(decryptedChar); // Thêm ký tự giải mã vào thông điệp
    }
    cout << endl;

    return strDecryptedMessage; // Trả về thông điệp đã giải mã
}
```

Hình 2.11 Hàm giải mã

b. Giao diện chương trình



Hình 2.12 Giao diện chương trình hệ mật RSA bằng C++

2.3.3.2 Cài đặt chương trình bằng c#

a. Thuật toán

```
using System;

0 references
class Program
{
    1 reference
    static bool IsPrime(int number)
    {

        if (number < 2)
            return false;
        for (int i = 2; i <= Math.Sqrt(number); i++)
        {
            if (number % i == 0)
                return false;
        }
        return true;
    }

    0 references
    static void Main(string[] args)
    {

        Console.WriteLine("Nhập một số: ");
        int input = int.Parse(Console.ReadLine());

        // Gọi hàm kiểm tra
        if (IsPrime(input))
            Console.WriteLine($"{input} là số nguyên tố.");
        else
            Console.WriteLine($"{input} không phải là số nguyên tố.");
    }
}
```

Hình 2.13 Hàm kiểm tra số nguyên tố bằng C#

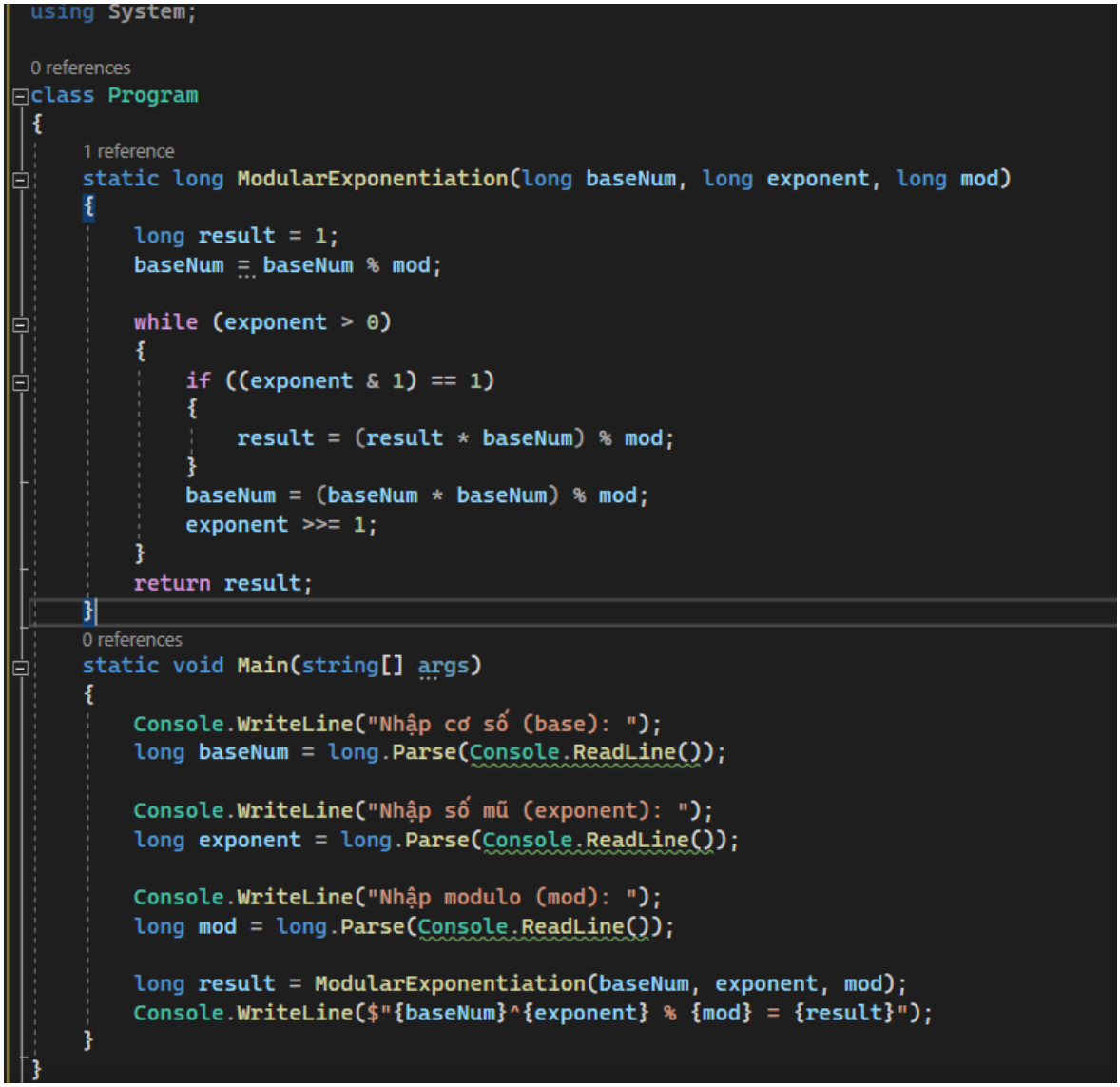
```
using System;

0 references
class Program
{
    1 reference
    static int GCD(int a, int b)
    {
        while (b != 0)
        {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }
    1 reference
    static bool AreRelativelyPrime(int num1, int num2)
    {
        return GCD(num1, num2) == 1;
    }
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Nhập số thứ nhất: ");
        int num1 = int.Parse(Console.ReadLine());

        Console.WriteLine("Nhập số thứ hai: ");
        int num2 = int.Parse(Console.ReadLine());

        if (AreRelativelyPrime(num1, num2))
            Console.WriteLine($"{num1} và {num2} là số nguyên tố cùng nhau.");
        else
            Console.WriteLine($"{num1} và {num2} không phải là số nguyên tố cùng nhau.");
    }
}
```

Hình 2.14 Hàm kiểm tra số nguyên tố cùng nhau bằng C#

```
using System;

0 references
class Program
{
    2 references
    static int ExtendedGCD(int a, int b, out int x, out int y)
    {
        if (b == 0)
        {
            x = 1;
            y = 0;
            return a;
        }
        int x1, y1;
        int gcd = ExtendedGCD(b, a % b, out x1, out y1);
        x = y1;
        y = x1 - (a / b) * y1;
        return gcd;
    }
    1 reference
    static int ModInverse(int a, int mod)
    {
        int x, y;
        int gcd = ExtendedGCD(a, mod, out x, out y);
        if (gcd != 1)
        {
            throw new ArgumentException("Số nghịch đảo không tồn tại vì GCD(a, mod) != 1.");
        }
        return (x % mod + mod) % mod;
    }
```

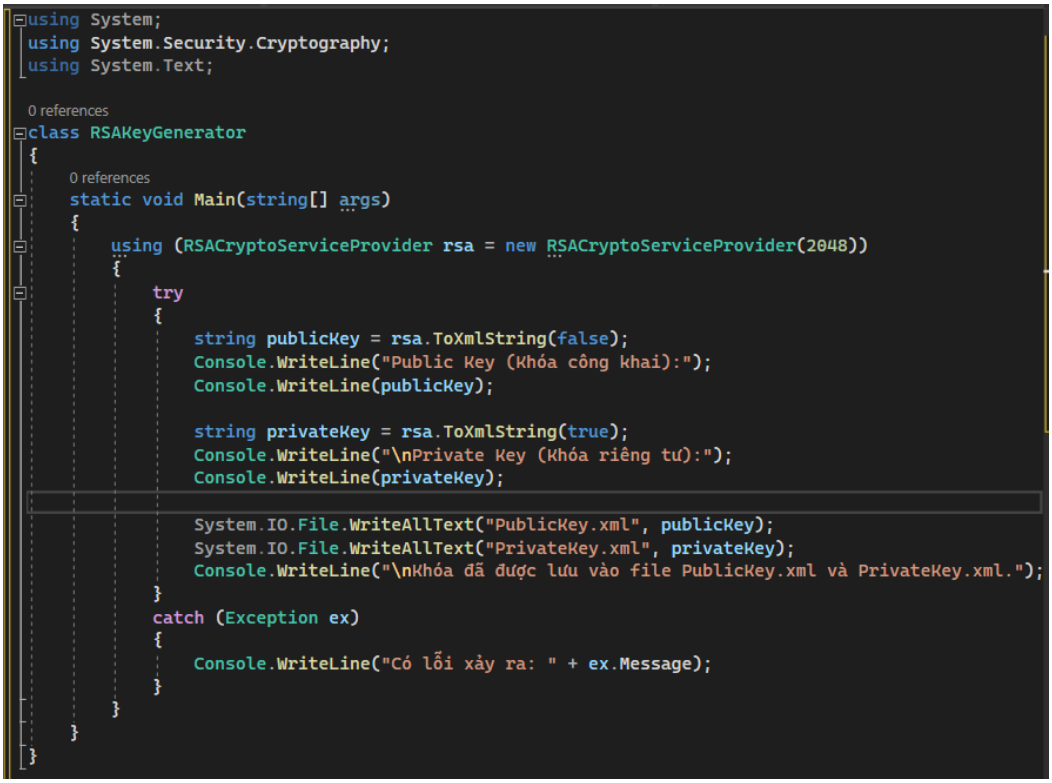
```
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Nhập a: ");
        int a = int.Parse(Console.ReadLine());

        Console.WriteLine("Nhập modulo: ");
        int mod = int.Parse(Console.ReadLine());

        try
        {
            int inverse = ModInverse(a, mod);
            Console.WriteLine($"Số nghịch đảo của {a} modulo {mod} là: {inverse}");
        }
        catch (ArgumentException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

Hình 2.15 Thuật toán Euclid mở rộng để tìm số nghịch đảo bằng C#



```
using System;

0 references
class Program
{
    1 reference
    static long ModularExponentiation(long baseNum, long exponent, long mod)
    {
        long result = 1;
        baseNum = baseNum % mod;

        while (exponent > 0)
        {
            if ((exponent & 1) == 1)
            {
                result = (result * baseNum) % mod;
            }
            baseNum = (baseNum * baseNum) % mod;
            exponent >>= 1;
        }
        return result;
    }
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Nhập cơ số (base): ");
        long baseNum = long.Parse(Console.ReadLine());

        Console.WriteLine("Nhập số mũ (exponent): ");
        long exponent = long.Parse(Console.ReadLine());

        Console.WriteLine("Nhập modulo (mod): ");
        long mod = long.Parse(Console.ReadLine());

        long result = ModularExponentiation(baseNum, exponent, mod);
        Console.WriteLine($"{baseNum}^{exponent} % {mod} = {result}");
    }
}
```

Hình 2.16 Thuật toán bình phương và nhân bằng C#



```
using System;
using System.Security.Cryptography;
using System.Text;

0 references
class RSAKeyGenerator
{
    0 references
    static void Main(string[] args)
    {
        using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(2048))
        {
            try
            {
                string publicKey = rsa.ToXmlString(false);
                Console.WriteLine("Public Key (Khóa công khai):");
                Console.WriteLine(publicKey);

                string privateKey = rsa.ToXmlString(true);
                Console.WriteLine("\nPrivate Key (Khóa riêng tư):");
                Console.WriteLine(privateKey);

                System.IO.File.WriteAllText("PublicKey.xml", publicKey);
                System.IO.File.WriteAllText("PrivateKey.xml", privateKey);
                Console.WriteLine("\nKhóa đã được lưu vào file PublicKey.xml và PrivateKey.xml.");
            }
            catch (Exception ex)
            {
                Console.WriteLine("Có lỗi xảy ra: " + ex.Message);
            }
        }
    }
}
```

Hình 2.17 Hàm sinh khóa

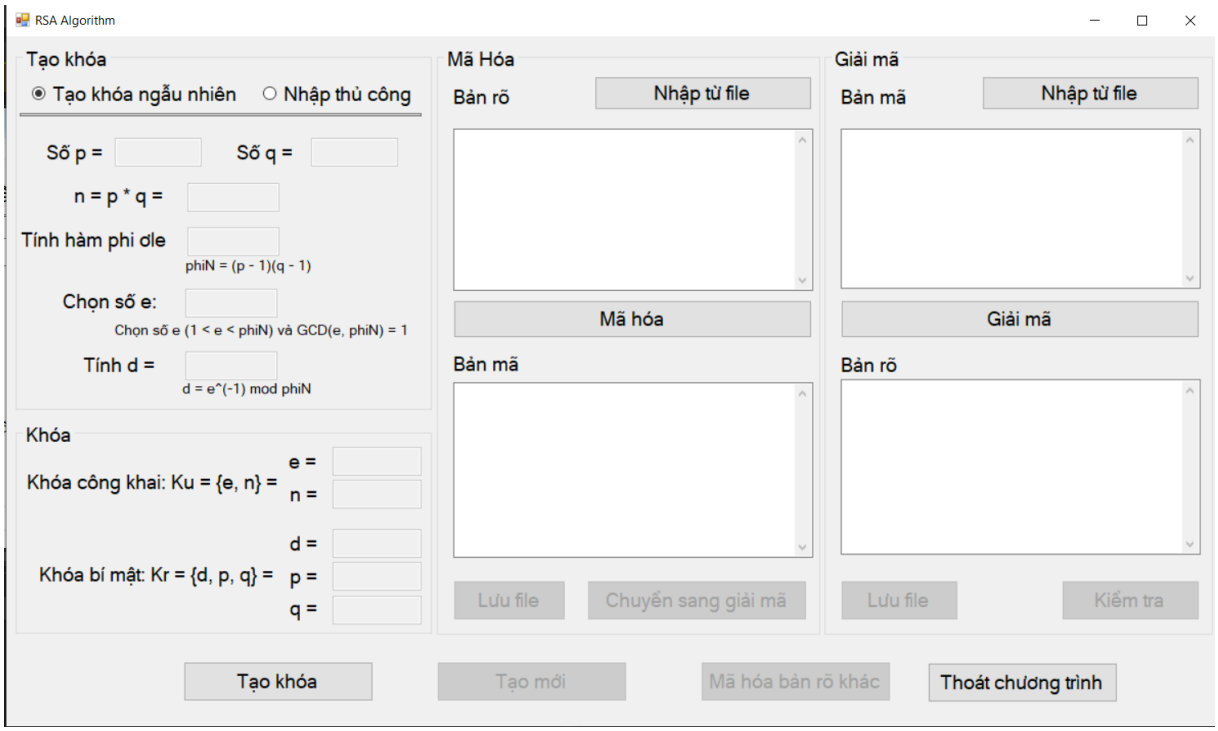
```
// Hàm mã hóa
1 reference
public string Encryption(string plainText, int soE, int soN)
{
    //Chuyển chuỗi đầu vào thành 1 mảng các byte sử dụng mã hóa Unicode
    byte[] encrypBytes = Encoding.Unicode.GetBytes(plainText);

    // Chuyển mảng byte thành một chuỗi base64-encoded
    string base64 = Convert.ToBase64String(encrypBytes);

    int[] charToUnicode = new int[base64.Length];

    //Lặp qua từng ký tự trong chuỗi base64 và gán giá trị Unicode tương ứng của ký tự vào mảng charToUnicode.
    for (int i = 0; i < base64.Length; i++)
    {
        charToUnicode[i] = (int)base64[i];
    }

    // Thực hiện tính mã hóa RSA thành các dãy các số với công thức: C = m^e mod n
    int[] encrypted_Array = new int[charToUnicode.Length];

    for (int i = 0; i != charToUnicode.Length; i++)
    {
        encrypted_Array[i] = RSA_Mod(charToUnicode[i], soN, soE);
    }

    // Mảng chứa các ký tự đã mã hóa:
    // Từ dãy các số ở trên ta ép kiểu nó về dạng ký tự tương ứng với giá trịn Unicode của mỗi phần tử
    string encrypStr = "";
    for (int i = 0; i < encrypted_Array.Length; i++)
    {
        encrypStr += (char)encrypted_Array[i];
    }

    // từ chuỗi ở trên ta chuyển nó thành một mảng byte bằng cách sử dụng mã hóa Unicode.
    byte[] data = Encoding.Unicode.GetBytes(encrypStr);

    //Chuyển đổi mảng byte thành một chuỗi base64-encoded
    string encryptedStr = Convert.ToBase64String(data);

    // trả về một chuỗi ký tự mã hóa RSA
    return encryptedStr;
}
```

Hình 2.18 Hàm thuật toán mã hóa sử dụng C#

```
// Hàm giải mã
1 reference
public string Decryption(string cipherText, int soD, int soN)
{
    // Chuyển đổi chuỗi mã hóa từ base64-encoded thành mảng byte
    byte[] decrypBytes = Convert.FromBase64String(cipherText);

    //// Chuyển đổi mảng byte thành chuỗi văn bản
    string decrypt = Encoding.Unicode.GetString(decrypBytes);

    // Lặp qua từng ký tự trong chuỗi decrypt và gán giá trị Unicode tương ứng của ký tự vào mảng b.
    int[] b = new int[decrypt.Length];
    for (int i = 0; i < decrypt.Length; i++)
    {
        b[i] = (int)decrypt[i];
    }

    // Từ mảng các số ở trên thực hiện tính toán theo công thức
    // giải mã RSA: m = c^d mod n
    // trong đó d: khóa bí mật, c: dạng unicode của ký tự muốn giải mã, n = p*q
    int[] m = new int[b.Length];
    for (int i = 0; i < b.Length; i++)
    {
        m[i] = RSA_Mod(b[i], soN, soD);
    }

    // Lặp qua từng phần tử trong mảng m và thêm ký tự tương ứng với giá trị Unicode của phần tử đó vào chuỗi decryptStr.
    string decryptStr = string.Empty;
    for (int i = 0; i < m.Length; i++)
    {
        decryptStr = decryptStr + (char)m[i];
    }

    //Chuyển đổi chuỗi decryptStr thành một mảng byte tương ứng
    byte[] data = Convert.FromBase64String(decryptStr);

    //Chuyển đổi mảng byte data thành chuỗi văn bản giải mã sử dụng mã unicode
    string decryptedStr = Encoding.Unicode.GetString(data);

    return decryptedStr;
}
```

Hình 2.19 Hàm thuật toán giải mã sử dụng C#

b. Giao diện chương trình



Hình 2.20 Giao diện màn hình chương trình demo bằng C#

CHƯƠNG 3 KẾT LUẬN VÀ BÀI HỌC KINH NGHIỆM

3.1 Kiến thức kỹ năng đã học được trong quá trình thực hiện đề tài.

Trong quá trình nghiên cứu và thực hiện đề tài "Tìm hiểu về hệ mật RSA và ứng dụng trong thực tế", nhóm chúng em đã đạt được những kết quả sau:

- **Hiểu rõ cơ sở lý thuyết:**
  - Nắm vững các khái niệm toán học nền tảng như số nguyên tố, hàm Euler, khái niệm modulo, và định lý Euler. Những kiến thức này là nền tảng để hiểu cách thức hoạt động của hệ mật RSA.
  - Tìm hiểu về cấu trúc và cách thức hoạt động của hệ mật mã bất đối xứng, đặc biệt là thuật toán RSA với các bước tạo khóa, mã hóa, và giải mã.
- **Ứng dụng thực tế:**
  - Nhóm đã triển khai hệ mã RSA bằng hai ngôn ngữ lập trình phổ biến là C++ và C#, giúp minh họa rõ ràng cách hoạt động của hệ thống.
  - Phân tích ưu nhược điểm của RSA trong bảo mật thông tin, ứng dụng của nó trong các lĩnh vực như thương mại điện tử và chữ ký số.
- **Xây dựng chương trình:**
  - Chương trình của nhóm đã thể hiện đầy đủ quy trình mã hóa và giải mã thông tin với các chức năng như sinh khóa, nhập dữ liệu, mã hóa thông điệp, và giải mã thông điệp.
  - Qua đó, khẳng định tính hiệu quả và ứng dụng cao của hệ mật RSA trong thực tế.

Những kết quả đạt được không chỉ dừng lại ở việc nghiên cứu lý thuyết mà còn là sự tiếp cận thực tế qua việc cài đặt và thử nghiệm thuật toán.

3.2 Bài học kinh nghiệm

- **Về mặt chuyên môn:**
  - Hiểu sâu hơn về các thuật toán và ứng dụng của mật mã học, đặc biệt là vai trò của RSA trong việc bảo mật thông tin.
  - Tăng cường khả năng áp dụng ngôn ngữ lập trình vào việc triển khai các thuật toán phức tạp.
- **Về kỹ năng làm việc nhóm:**
  - Kỹ năng phân chia công việc: Nhóm đã biết cách phân công nhiệm vụ một cách hợp lý, tận dụng tối đa năng lực của từng thành viên.
  - Kỹ năng giao tiếp và hợp tác: Thường xuyên trao đổi, thảo luận để giải quyết các vấn đề gặp phải trong quá trình nghiên cứu.
- **Khắc phục khó khăn:**
  - Đối mặt với thách thức trong việc xử lý các số nguyên lớn và tối ưu hóa thuật toán, nhóm đã học được cách tìm kiếm và sử dụng các thư viện hỗ trợ trong lập trình như <System.Numerics> trong C# hay <cmath> trong C++.
  - Học cách xử lý lỗi và tối ưu mã nguồn để đảm bảo chương trình hoạt động chính xác và hiệu quả.

3.3 Đề xuất về tính khả thi của chủ đề nghiên cứu, những thuận lợi, khó khăn

- **Hiểu rõ cơ sở lý thuyết:**

- Nắm vững các khái niệm toán học nền tảng như số nguyên tố, hàm Euler, khái niệm modulo, và định lý Euler. Những kiến thức này là nền tảng để hiểu cách thức hoạt động của hệ mật RSA.
- Tìm hiểu về cấu trúc và cách thức hoạt động của hệ mật mã bất đối xứng, đặc biệt là thuật toán RSA với các bước tạo khóa, mã hóa, và giải mã.

- **Ứng dụng thực tế:**
  - Nhóm đã triển khai hệ mã RSA bằng hai ngôn ngữ lập trình phổ biến là C++ và C#, giúp minh họa rõ ràng cách hoạt động của hệ thống.
  - Phân tích ưu nhược điểm của RSA trong bảo mật thông tin, ứng dụng của nó trong các lĩnh vực như thương mại điện tử và chữ ký số.
- **Xây dựng chương trình:**
  - Chương trình của nhóm đã thể hiện đầy đủ quy trình mã hóa và giải mã thông tin với các chức năng như sinh khóa, nhập dữ liệu, mã hóa thông điệp, và giải mã thông điệp.
  - Qua đó, khẳng định tính hiệu quả và ứng dụng cao của hệ mật RSA trong thực tế.

TÀI LIỆU THAM KHẢO

[1]. Nguyễn Xuân Dũng, Bảo mật thông tin – Mô Hình và ứng dụng, NXB Thống Kê, 2009.

[2]. Bùi Doãn Khanh, Nguyễn Đình Thúc, Mã hóa thông tin – Lý thuyết và ứng dụng, NXB Lao động Xã hội, 2011.

[3]. William Stallings, Cryptography and Network Security Principles and Practices, Fourth Edition, Prentice Hall, 2005.

[4]. Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[5]. Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition, John Wiley & Sons, 1996.