ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÀI TẬP LỚN
MÔN MẬT MÃ VÀ AN NINH MẠNG
ĐỀ TÀI

HIỆN THỰC HỆ MÃ RSA BẰNG C++

Lớp L01 - HK 241

Giảng viên hướng dẫn: ThS. Nguyễn Cao Đạt

Sinh viên thực hiện	MSSV
Hoàng Ngô Thiên Phúc	2212612
Nguyễn Ngọc Phú	2212588
Lê Thanh Phú	2212581
Hồ Tây Nguyên	2036076

Thành phố Hồ Chí Minh - 2024

Mục lục

Danh sách thành viên và đóng góp

Sinh viên thực hiện	**MSSV**	**Đóng góp**
Hoàng Ngô Thiên Phúc	2212612	100%
Nguyễn Ngọc Phú	2212588	100%
Lê Thanh Phú	2212581	100%
Hồ Tây Nguyên	2036076	100%

Chương 1

Giới thiệu

RSA là một hệ mã hóa bất đối xứng được phát minh vào năm 1977 bởi Ron Rivest, Adi Shamir và Leonard Adleman. Hệ mã này sử dụng hai khóa: một khóa công khai để mã hóa dữ liệu và một khóa riêng tư để giải mã. RSA dựa trên bài toán phân tích số nguyên lớn thành các thừa số nguyên tố, một bài toán rất khó giải quyết, từ đó đảm bảo tính bảo mật của hệ thống. RSA không chỉ được sử dụng trong việc mã hóa dữ liệu mà còn đóng vai trò quan trọng trong việc xác thực thông tin thông qua chữ ký số. Mặc dù RSA có tính bảo mật cao, nhưng tốc độ của nó thường chậm hơn so với các hệ mã hóa đối xứng, khiến nó thường được sử dụng kết hợp với các phương pháp mã hóa khác trong thực tế.

Trong bài tập này, nhóm sẽ thực hiện hệ mã RSA bằng ngôn ngữ lập trình C++ sử dụng các thuật toán cơ bản của mật mã học. Cụ thể là sử dụng thuật toán Miller-Rabin để kiểm tra tính nguyên tố của các số lớn, giúp tạo ra các số nguyên tố ngẫu nhiên cần thiết cho hệ mã RSA. Áp dụng Thuật toán Euclid mở rộng để tính toán khóa giải mã, qua đó đảm bảo rằng các phép toán như tìm ước số chung lớn nhất (GCD) và nghịch đảo modulo được thực hiện chính xác. Thư viện NTL (A Library For Doing Number Theory) được sử dụng để xử lý số lớn. Mục tiêu của bài tập là không chỉ hiểu rõ các khái niệm cơ bản về RSA mà còn rèn luyện kỹ năng lập trình và xử lý các bài toán số học phức tạp trong mật mã học.

Chương 2

Cơ sở lý thuyết

I Thuật toán RSA

RSA là một thuật toán mã bất đối xứng phổ biến, dựa trên các tính chất của số học mô-đun. Thuật toán này bao gồm ba giai đoạn chính: tạo khóa (key generation), mã hóa (encryption) và giải mã (decryption).

1 Các bước hiện thực RSA

Bước 1: Tạo khóa

1. Chọn hai số nguyên tố lớn p và q với $p \neq q$, lựa chọn ngẫu nhiên và độc lập.
2. Tính: $n = p \times q$.
3. Tính giá trị hàm số Euler $\varphi(n) = (p-1)(q-1)$.
4. Chọn một số tự nhiên e sao cho $1 < e < \varphi(n)$ và e là số nguyên tố cùng nhau với $\varphi(n)$.
5. Tính d sao cho $d \times e \equiv 1 \pmod{\varphi(n)}$.

Bước 2: Mã hóa

Để mã hóa c từ thông điệp m, ta thực hiện: $c = m^e \mod n$.

Bước 3: Giải mã

Để giải mã, ta thực hiện: $m = c^d \mod n$.

2 Đặc điểm và tính chất

- **Bảo mật**: Độ an toàn của hệ thống RSA dựa trên 2 vấn đề của toán học: bài toán phân tích ra thừa số nguyên tố các số nguyên lớn và bài toán RSA.
- **Ứng dụng**: Chữ ký số, bảo mật email, HTTPS.

II Số học modulo, thuật toán Euclid và Euclid mở rộng

1 Số học modulo

1.1 Khái niệm cơ bản

Định nghĩa: Số học modulo là hệ thống số học mà trong đó các phép toán được thực hiện trên phần dư của phép chia. Ký hiệu:

$$a \mod n = r \quad \text{với } a = nq + r \text{ và } 0 \leq r < n.$$

Trong đó: a là số bị chia. n là số chia (modulus). r là phần dư.

Tính chất:

1. $a \equiv b \pmod{n} \iff (a - b)$ chia hết cho n.
2. Nếu $a \equiv b \pmod{n}$ và $c \equiv d \pmod{n}$, thì:
   $a + c \equiv b + d \pmod{n}$,
   $a - c \equiv b - d \pmod{n}$,
   $a \times c \equiv b \times d \pmod{n}$.

1.2 Ứng dụng

- Hệ thống mã hóa (như RSA).
- Lý thuyết đồng dư trong toán học.
- Giải các bài toán phân nhóm hoặc chia phần dư.

2 Thuật toán Euclid

2.1 Khái niệm

Thuật toán Euclid là phương pháp hiệu quả để tính **ước số chung lớn nhất (GCD)** của hai số nguyên a và b.

Ý tưởng cơ bản:

$$\gcd(a, b) = \gcd(b, a \mod b).$$

Thuật toán tiếp tục lặp cho đến khi $b = 0$, khi đó:

$$\gcd(a, 0) = a.$$

2.2 Các bước thực hiện

1. Bắt đầu với hai số a và b ($a > b$).
2. Tính phần dư: $r = a \mod b$.
3. Gán $a = b$ và $b = r$.
4. Lặp lại cho đến khi $b = 0$.
5. Kết quả $\gcd(a, b)$ là a ở bước cuối cùng.

2.3 Ví dụ minh họa

Tìm $\gcd(48, 18)$:

$$\begin{aligned} 48 &\mod 18 = 12 \quad \text{(bước 1)} \\ 18 &\mod 12 = 6 \quad \text{(bước 2)} \\ 12 &\mod 6 = 0 \quad \text{(bước 3)} \end{aligned}$$

Kết quả: $\gcd(48, 18) = 6$.

3 Thuật toán Euclid mở rộng

3.1 Khái niệm

Thuật toán Euclid mở rộng không chỉ tìm $\gcd(a, b)$, mà còn xác định các hệ số x và y sao cho:

$$ax + by = \gcd(a, b).$$

Đây là nền tảng để tính **nghịch đảo modulo** trong số học modulo.

3.2 Quy trình

1. Áp dụng thuật toán Euclid để tìm $\gcd(a, b)$.
2. Lưu lại các phép chia trong quá trình:

$$a = bq_1 + r_1, \quad b = r_1q_2 + r_2, \ldots, r_{n-1} = r_nq_{n+1} + 0.$$

3. Từ các phép chia này, sử dụng phép thay thế ngược để tìm x và y.

3.3 Ví dụ minh họa

Tìm $\gcd(30, 12)$ và các hệ số x, y:

1. Áp dụng thuật toán Euclid:

$$30 = 12 \cdot 2 + 6, \quad 12 = 6 \cdot 2 + 0. \quad \gcd(30, 12) = 6.$$

2. Thay ngược:

$$6 = 30 - 12 \cdot 2.$$

Vậy $x = 1$, $y = -2$, và:

$$30 \cdot 1 + 12 \cdot (-2) = 6.$$

3.4 Ứng dụng của Euclid mở rộng

1. **Tính nghịch đảo modulo**: Nếu $\gcd(a, n) = 1$, sử dụng Euclid mở rộng để tìm x sao cho:

$$ax \equiv 1 \pmod{n}.$$

2. **Hệ thống mật mã RSA**: Tính khóa giải mã d trong RSA bằng cách giải phương trình:

$$ed \equiv 1 \pmod{\phi(n)}.$$

III Thuật toán Miller-Rabin

Thuật toán Miller-Rabin là một thuật toán xác suất dùng để kiểm tra tính nguyên tố của một số nguyên N. Nó được đề xuất đầu tiên bởi Gary L. Miller như một thuật toán tất định, dựa trên giả thuyết Riemann tổng quát, và Michael O. Rabin đã sửa chữa nó thành một thuật toán xác suất.

1 Cơ sở lý thuyết

Nguyên tố và hợp số

Một số dương $n > 1$ được gọi là số nguyên tố nếu nó chỉ chia hết cho 1 và chính nó. Nếu không, nó là hợp số.

Định lý Fermat

Định lý này cho biết rằng nếu p là số nguyên tố và a là một số nguyên dương sao cho $1 \leq a < p$ thì:

$$a^{p-1} \equiv 1 \pmod{p}$$

Định lý này cung cấp một cách để kiểm tra xem n có thể là số nguyên tố hay không bằng cách tính giá trị $a^{n-1} \mod n$. Nếu kết quả khác 1, thì n không phải là số nguyên tố. Tuy nhiên, chỉ sử dụng định lý Fermat là không đủ vì có số Carmichael, tức là những hợp số n thỏa mãn $a^{n-1} \equiv 1 \pmod{n}$ với mọi a nguyên tố cùng nhau với n.

Phân tích $n-1$ thành dạng $2^s \cdot d$

Một số đặc điểm quan trọng của thuật toán Miller-Rabin là nó không chỉ dựa vào điều kiện $a^{n-1} \equiv 1 \pmod{n}$, mà còn khai thác thêm cấu trúc $n-1$. Cụ thể: Cho $n-1$ có dạng phân tích:

$$n - 1 = 2^s \cdot d$$

trong đó d là số lẻ. Phép phân tích này giúp kiểm tra n sâu hơn thông qua các điều kiện mạnh hơn.

2 Kiểm tra Miller-Rabin

Thuật toán kiểm tra Miller-Rabin thực hiện như sau:

1. **Phân tích $n-1$:** Tìm s và d sao cho $n - 1 = 2^s \cdot d$, với d là số lẻ.
2. **Chọn ngẫu nhiên một số a:** a được chọn ngẫu nhiên trong khoảng $[2, n-2]$.
3. **Kiểm tra điều kiện:** Tính:
$$x = a^d \mod n$$
Nếu $x \equiv 1 \pmod{n}$ hoặc $x \equiv n-1 \pmod{n}$ thì n có thể là số nguyên tố. Nếu không, tính dãy:
$$a^{2^r \cdot d} \mod n \quad \text{cho } r = 1, 2, \ldots, s-1$$
Nếu tồn tại r sao cho $a^{2^r \cdot d} \equiv n-1 \pmod{n}$ thì n có thể là số nguyên tố. Nếu tất cả thất bại thì n là hợp số.
4. **Lặp lại với nhiều giá trị a:** Thuật toán được thực hiện nhiều lần với các giá trị a khác nhau để giảm xác suất sai lầm.

3 Tính chất xác suất

Nếu n là hợp số, thì xác suất có một số a bất kỳ không phát hiện được n là hợp số là không quá $\frac{1}{4}$. Do đó, bằng cách lặp lại bài kiểm tra với nhiều giá trị a, xác suất sai lầm có thể giảm

đến mức rất nhỏ.

4 Ưu và nhược điểm

- **Ưu điểm:** Thuật toán Miller-Rabin rất nhanh và hiệu quả, phù hợp để kiểm tra tính nguyên tố của các số lớn.
- **Nhược điểm:** Đây là thuật toán xác suất, tức là nó không đảm bảo chắc chắn 100% n là số nguyên tố. Tuy nhiên, với đủ số lần kiểm tra, xác suất sai sót có thể giảm đến mức không đáng kể.

5 Ví dụ minh họa

Kiểm tra $n = 561$ (một số Carmichael) với $a = 2$:

1. Ta có:
$$n - 1 = 560 = 2^4 \cdot 35.$$
2. Tính:
$$2^{35} \mod 561 = 263 \neq 1 \quad \text{và} \quad 263 \neq 560$$
3. Tính tiếp:
$$2^{70} \mod 561 = 166$$
$$2^{140} \mod 561 = 67$$
$$2^{280} \mod 561 = 1$$
4. Không có giá trị nào bằng 560. Vậy $n = 561$ là hợp số.

6 Kết luận

Miller-Rabin là công cụ quan trọng trong lý thuyết số và mật mã học, đặc biệt khi kiểm tra tính nguyên tố cho các số lớn.

Chương 3

Phân tích và thiết kế

I Phân tích yêu cầu

1 Các chức năng chính

1. Tìm số nguyên tố lớn.
2. Tìm ước chung lớn nhất.
3. Tính khóa giải mã d khi biết e, q, n.
4. Tạo bộ khóa (khóa công khai e, n và khóa riêng d, n).
5. Mã hóa thông điệp bằng khóa công khai.
6. Giải mã thông điệp bằng khóa riêng.

2 Ràng buộc

1. Không sử dụng các thư viện hoặc phương thức hiện thực RSA có sẵn.
2. Chỉ được sử dụng thư viện hỗ trợ tính toán số nguyên lớn như NTL hoặc GMP.

II Phân tích hệ thống thành các module

1 Module Sinh số nguyên tố lớn

1. Nhận đầu vào: số lượng bit.
2. Sinh một số ngẫu nhiên lớn và kiểm tra tính nguyên tố.

2 Module Tính GCD

Nhận vào 2 số nguyên tố lớn, hiện thực thuật toán Euclid mở rộng để tính $GCD\ (a,\ b)$.

3 Module Tính khóa giải mã d

Sử dụng thuật toán Euclid mở rộng để tìm d, thỏa mãn:

$$d \times e \equiv 1\ (\ mod\ (p-1) \times (q-1)).$$

4 Module Sinh bộ khóa

1. Sinh hai số nguyên tố lớn p, q. Tính $n = p \times q$ và $\phi(n) = (p-1) \times (q-1)$.
2. Chọn e sao cho $\gcd(e, \phi(n)) = 1$.
3. Tính khóa giải mã d.

5 Module Mã hóa thông điệp

Nhận đầu vào: thông điệp M, khóa công khai e,n. Tính $C = M^e\ mod\ n$.

6 Module Giải mã thông điệp

1. Nhận đầu vào: thông điệp mã hóa C, khóa giải mã d, n.
2. Tính $M = C^d \mod n$.

III Mô tả luồng hoạt động

1. Người dùng nhập thông điệp để mã hóa
2. Hệ thống sinh ra số nguyên tố lớn p, q theo số bit yêu cầu
3. Tính $n, \phi(n)$ và tạo khóa $e,\ d$.
4. Chương trình mã hóa thông điệp thành cipher text
5. Chương trình giải mã trả về thông điệp gốc cho người dùng

Chương 4

Hiện thực và đánh giá

I Hiện thực thuật toán RSA

- Chương trình gồm các file có cấu trúc như sau:
  1. `main.hpp`: Khai báo thư viện gồm NTL, iostream, string và namespace NTL, std.
  2. `main.cpp`: Triển khai mã hoá và giải mã.
  3. `RSA.hpp`: Khai báo các hàm RSA.
  4. `RSA.cpp`: Hiện thực các hàm RSA.
- Các hàm trong `RSA.hpp` và `RSA.cpp` cùng với nhiệm vụ chi tiết:
  1. `ZZ stringToNumber(string str);`
  2. `string numberToString(ZZ num);`
  3. `ZZ InverseMod(ZZ a, ZZ m);`
  4. `ZZ power_mod(ZZ base, ZZ exp, ZZ mod);`
  5. `bool MillerRabinTest(ZZ to_test);`
  6. `ZZ random_prime(int bit_length);`
  7. `ZZ gcd(ZZ a, ZZ b);`
  8. `void generate_keypair(ZZ& n, ZZ& e, ZZ& d, int keyLength);`
  9. `ZZ encrypt(const ZZ& m, const ZZ& e, const ZZ& n);`
  10. `ZZ decrypt(const ZZ& c, const ZZ& d, const ZZ& n);`

1 Chuyển chuỗi sang số và ngược lại

1.1. ZZ stringToNumber(string str):

Hàm này chuyển một chuỗi ký tự thành một số nguyên lớn (ZZ). Mỗi ký tự trong chuỗi được chuyển đổi thành một số theo mã ASCII, và sau đó các mã ASCII này được kết hợp lại thành một số nguyên duy nhất.

- Ký tự đầu tiên được lấy và chuyển đổi thành số.
- Các ký tự còn lại được nhân với 128 (để "dịch" một byte) và cộng vào giá trị tổng, tạo ra một số đại diện cho toàn bộ chuỗi.

1.2. String numberToString(ZZ num):

Hàm này thực hiện phép chuyển đổi ngược lại, tức là chuyển một số nguyên lớn thành một chuỗi ký tự.

- Sử dụng phép toán modulo 128, mỗi "byte" được trích xuất từ số nguyên và chuyển thành ký tự tương ứng.
- Các ký tự được ghép lại thành chuỗi, sử dụng `static_cast<unsigned char>` để đảm bảo rằng mỗi số nguyên tương ứng với một ký tự ASCII hợp lệ.

2 Tính nghịch đảo Modulo

Hàm **ZZ InverseMod(ZZ a, ZZ m)** tính nghịch đảo modulo của một số a theo modulo m, tức là tìm x sao cho:

$$a \times x \equiv 1 \pmod{m}$$

Thuật toán sử dụng **Thuật toán Euclid mở rộng** để tìm nghịch đảo. Quá trình này gồm các phép chia liên tiếp và cập nhật các giá trị của x sao cho cuối cùng ta có thể tìm được x thỏa mãn điều kiện trên. Nếu kết quả của x_1 là âm, ta cộng thêm m_0 vào để đảm bảo x_1 là số dương.

3 Lũy thừa Modulo

Hàm **ZZ power_mod(ZZ base, ZZ exp, ZZ mod)** tính $base^{exp} \mod mod$ sử dụng phương pháp **lũy thừa nhanh** (exponentiation by squaring). Phương pháp này rất hiệu quả trong việc tính toán lũy thừa với các số lớn và giảm thiểu độ phức tạp tính toán. Cụ thể:

- Nếu số mũ exp là lẻ, ta nhân kết quả với base.
- Nếu exp là chẵn, ta bình phương base lên và chia mũ cho 2 cho đến khi mũ trở về 0.

4 Kiểm tra số nguyên tố bằng phương pháp Miller-Rabin

Hàm **bool MillerRabinTest(ZZ to_test)** kiểm tra số nguyên tố ngẫu nhiên dựa trên phương pháp Miller-Rabin. Hàm này sử dụng biểu diễn:

$$to\_test - 1 = 2^s \times d$$

để chia $to\_test - 1$ thành một phần lẻ d và một phần số mũ 2 s, từ đó kiểm tra tính nguyên tố của to_test thông qua một số lần kiểm tra ngẫu nhiên. Nếu kết quả trả về không chứng minh số nguyên tố, số cần kiểm tra sẽ bị loại bỏ.

5 Sinh số nguyên tố ngẫu nhiên

Hàm **ZZ random_prime(int bit_length)** sinh ra một số nguyên tố ngẫu nhiên có độ dài bit cho trước. Sử dụng hàm `MillerRabinTest` để xác minh rằng số được sinh ra là nguyên tố. Nếu số không phải nguyên tố, hàm sẽ tiếp tục sinh số khác cho đến khi tìm được một số nguyên tố hợp lệ.

6 Tính ước chung lớn nhất

Hàm **ZZ gcd(ZZ a, ZZ b)** tính ước chung lớn nhất của hai số a và b sử dụng **thuật toán Euclid**. Thuật toán này lặp lại việc chia và lấy dư cho đến khi phần dư bằng 0, lúc đó giá trị còn lại chính là GCD.

7 Tạo cặp khóa RSA

Hàm **void generate_keypair(ZZ& n, ZZ& e, ZZ& d, int keyLength)** tạo ra cặp khóa RSA. Nó thực hiện các bước sau:

- Sinh hai số nguyên tố p và q.
- Tính $n = p \times q$.
- Tính $\varphi(n) = (p-1) \times (q-1)$, là giá trị dùng để tính khóa riêng.
- Chọn e là một số nguyên tố nhỏ, điển hình là 65537 (một giá trị phổ biến trong RSA).
- Tính khóa riêng d sao cho:

$$e \times d \equiv 1 \pmod{\varphi(n)}$$

8 Mã hóa và giải mã

Cả mã hóa và giải mã đều sử dụng phương thức lũy thừa modulo (`power_mod`) để xử lý các phép toán với số nguyên lớn.

Hàm mã hóa **ZZ encrypt(const ZZ& m, const ZZ& e, const ZZ& n)** theo công thức:

$$c = m^e \mod n$$

Hàm giải mã **ZZ decrypt(const ZZ& c, const ZZ& d, const ZZ& n)** theo công thức:

$$m = c^d \mod n$$

9 Mô tả User Interface cho Chương Trình Mã Hóa và Giải Mã RSA

Tổng quan chương trình

Chương trình cung cấp giao diện dòng lệnh (CLI) để thực hiện các chức năng mã hóa và giải mã thông điệp sử dụng thuật toán RSA. Người dùng có thể lựa chọn sinh cặp khóa RSA mới hoặc sử dụng khóa đã có sẵn.

Menu chính

Khi khởi động chương trình, giao diện hiển thị menu chính với ba tùy chọn:

```
Menu Mã Hóa RSA:
1. Mã Hóa Thông Điệp
2. Giải Mã Thông Điệp
3. Thoát
Chọn:
```

- **Người dùng nhập lựa chọn:** Chọn `1` để mã hóa thông điệp, `2` để giải mã thông điệp, hoặc `3` để thoát chương trình.
- **Xử lý lựa chọn không hợp lệ:** Nếu nhập sai, chương trình sẽ thông báo và yêu cầu nhập lại.

Mã hóa thông điệp

Khi người dùng chọn mã hóa thông điệp (tùy chọn 1), giao diện sẽ dẫn dắt qua các bước sau:

1. **Nhập thông điệp cần mã hóa:**

   ```
   Nhập thông điệp để mã hóa:
   ```

   Người dùng nhập chuỗi văn bản cần mã hóa.

2. **Chọn độ dài khóa:**

`Nhập độ dài khóa (khuyến nghị 1024, 2048, 4096):`

- Chỉ chấp nhận các giá trị từ 1024 đến 4096.
- Thông báo lỗi nếu nhập độ dài khóa không hợp lệ.

3. **Sinh khóa hoặc sử dụng khóa đã có:**

   `Sinh cặp khóa mới? (y/n):`

   - **Y:** Chương trình sẽ tự động sinh cặp khóa RSA và lưu vào hai file `public.key` và `private.key`.
   - **N:** Người dùng nhập đường dẫn tới file khóa công khai:

     `Nhập đường dẫn file public key (Enter để dùng mặc định):`

     Nếu không nhập, chương trình mặc định sử dụng `public.key`.

4. **Nhập đường dẫn lưu kết quả mã hóa:**

   `Nhập đường dẫn lưu file cipher (Enter để dùng mặc định cipher.txt):`

   Người dùng có thể đặt tên file tùy ý hoặc để trống để dùng tên mặc định.

5. **Hiển thị thông điệp mã hóa:**
   Sau khi mã hóa xong, chương trình thông báo kết quả:

   ```
   Thông điệp đã được mã hóa và lưu tại [đường dẫn].
   Thông điệp mã hóa: [nội dung mã hóa]
   ```

Giải mã thông điệp

Khi người dùng chọn giải mã thông điệp (tùy chọn 2), giao diện thực hiện các bước sau:

1. **Nhập đường dẫn file khóa riêng tư:**

   `Nhập đường dẫn file private key (Enter để dùng mặc định):`

   Nếu không nhập, chương trình mặc định sử dụng `private.key`.

2. **Kiểm tra tồn tại file khóa và file cipher:**
   - Nếu file không tồn tại, chương trình thông báo lỗi và kết thúc quá trình giải mã.
3. **Nhập đường dẫn file chứa dữ liệu mã hóa:**

   ```
   Nhập đường dẫn file cipher (Enter để dùng mặc định cipher.txt):
   ```

4. **Hiển thị thông điệp giải mã:**
   Sau khi giải mã xong:

   ```
   Thông điệp giải mã (số): [giá trị số]
   Thông điệp gốc: [chuỗi gốc]
   ```

Xử lý lỗi và thông báo

- **Lỗi mở file:** Chương trình hiển thị thông báo cụ thể nếu không thể mở hoặc đọc/ghi file.
- **Lỗi nhập liệu:** Các thông báo rõ ràng khi người dùng nhập sai hoặc thông tin không hợp lệ.
- **Cảnh báo bảo mật:** Gợi ý lưu trữ an toàn các khóa RSA sau khi tạo.

Kết luận

Giao diện của chương trình được thiết kế đơn giản và trực quan, đảm bảo người dùng có thể thực hiện các thao tác mã hóa và giải mã một cách dễ dàng. Từng bước trong quy trình đều đi kèm với hướng dẫn và thông báo rõ ràng, giúp giảm thiểu sai sót khi sử dụng.

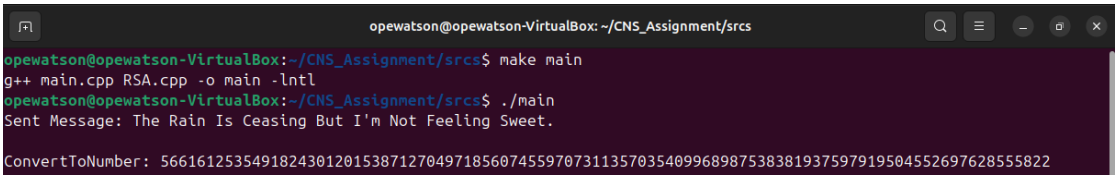
II Đánh giá

1 Chuyển chuỗi sang số và ngược lại

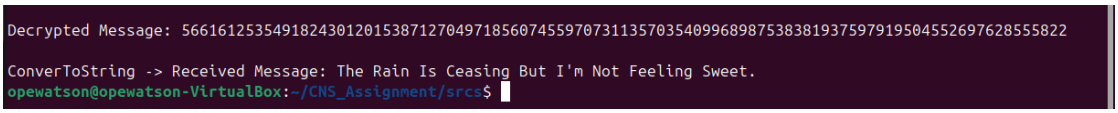
Hàm **ZZ stringToNumber(string str)** và **ZZ numberToString (ZZ num)** thực hiện quá trình chuyển số sang chuỗi và ngược lại theo quy tắc mã hoá UTF-8 chính xác kể cả kí tự đặc biệt.

2 Tính nghịch đảo Modulo

Hàm **ZZ InverseMod(ZZ a, ZZ m)** tính chính xác ngịch đảo modulo của các số tương đối lớn và rất lớn. Thử lại bằng công cụ tính Inverse Mod: https://planetcalc.com/3311/



Hình 4.1: Chuyển chuỗi sang số.



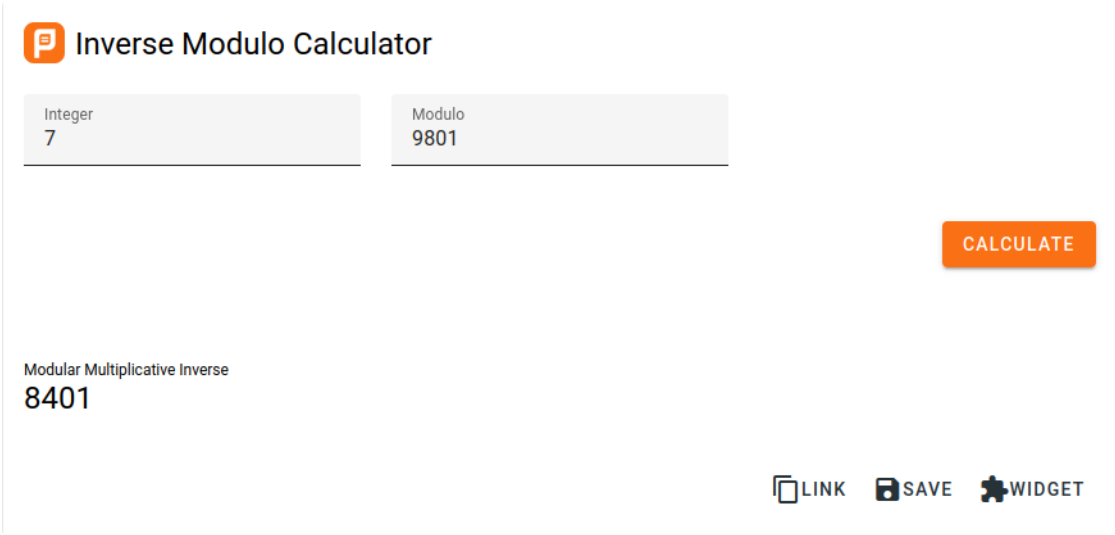
Hình 4.2: Chuyển số về lại chuỗi.

3 Sinh số nguyên tố ngẫu nhiên

Hàm **ZZ random_prime(int bit_length)** sinh ra số ngẫu nhiên >512 bits có độ chính xác cao và được kiểm tra bằng công cụ Prime Number Checker: https://www.numberempire.com/primenumbers.php

```
n = 10000
phi_n = 9801
e = 7
d = 8401
```

Hình 4.3: Tính Inverse Mod.

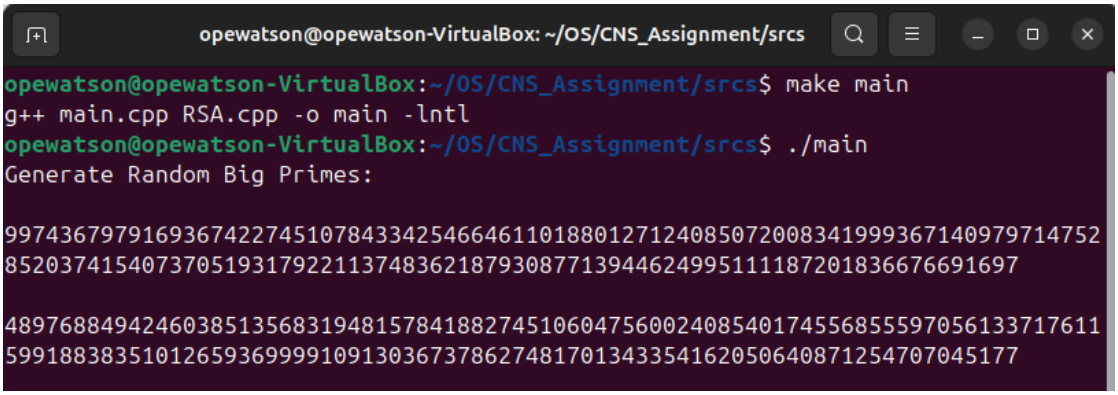


Hình 4.4: Thử lại bằng công cụ khác.

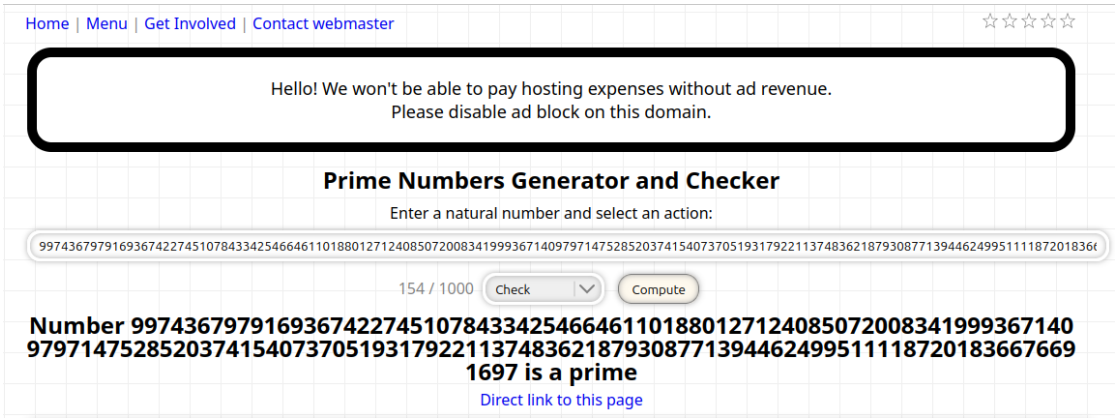
4 Thử nghiệm chương trình

Chương trình thực hiện thuật toán RSA và cho kết quả chính xác. Thời gian chạy của chương trình tỷ lệ thuận với độ dài khóa. Cụ thể, với khóa có độ dài 512 bits, thời gian chạy là 0.058 giây, trong khi với khóa 4096 bits, thời gian chạy tăng lên đến 22.555 giây.

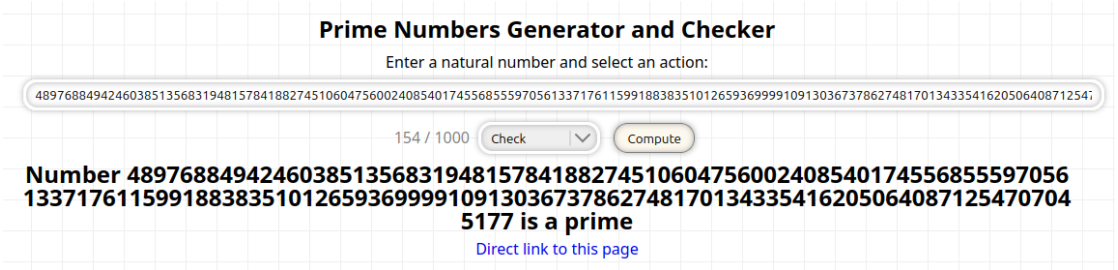
- Với khóa 512 bits, thời gian chạy là 0m0.058s.
- Với khóa 4096 bits, thời gian chạy là 0m22.555s.



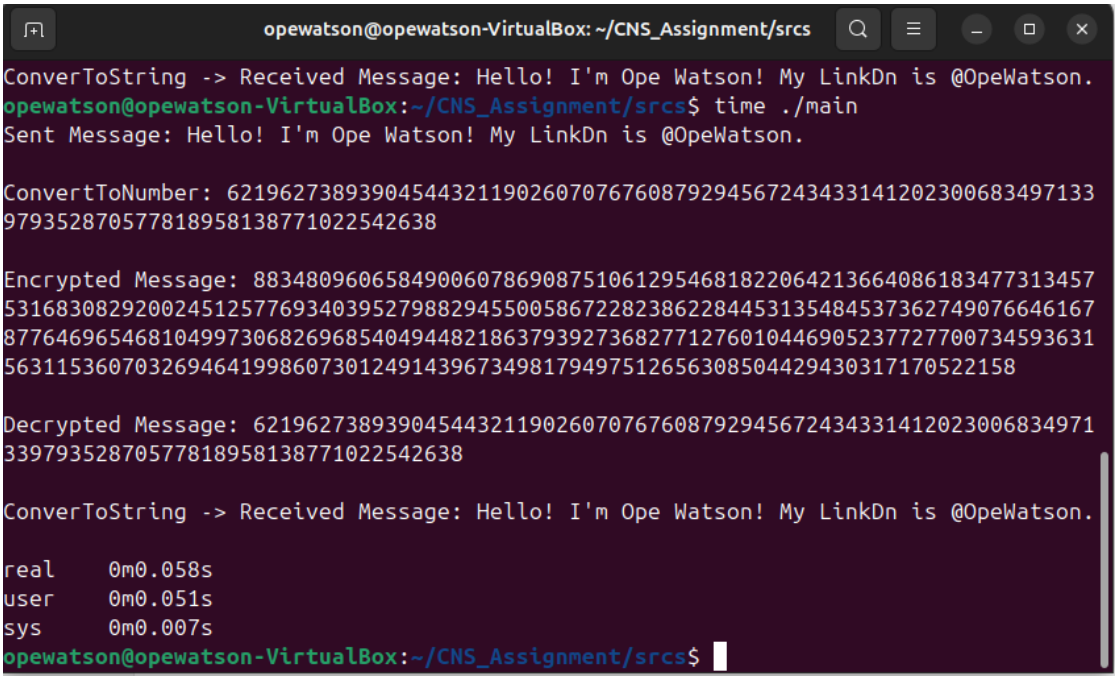
Hình 4.5: Sinh số nguyên tố kiểm tra bằng Miller Rabbin.



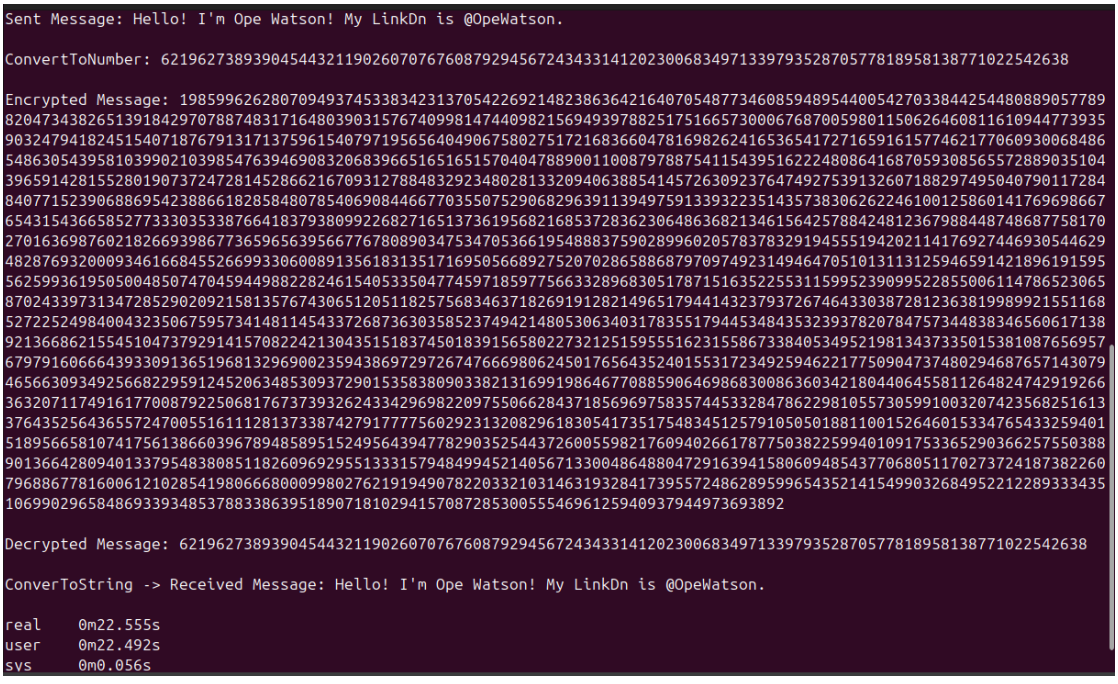
Hình 4.6: Kiểm tra bằng công cụ khác 1.



Hình 4.7: Kiểm tra bằng công cụ khác 2.



Hình 4.8: Chạy thử với độ dài khoá 512 bits.



Hình 4.9: Chạy thử với độ dài khoá 4096 bits.

Chương 5

Kết luận

Qua quá trình thực hiện bài tập lớn, chúng em đã có cơ hội khám phá và tìm hiểu sâu sắc về mã hóa khóa công khai, đặc biệt là hệ mã RSA, cũng như các phương pháp mã hóa công khai khác. Thông qua việc nghiên cứu và triển khai thực tế, chúng em không chỉ củng cố kiến thức lý thuyết mà còn rèn luyện kỹ năng hiện thực hóa hệ mã RSA bằng ngôn ngữ C++. Đây là một trải nghiệm vô cùng quý báu, giúp chúng em hiểu rõ hơn về các nguyên lý mã hóa và ứng dụng của chúng trong lĩnh vực bảo mật thông tin.

Trong suốt quá trình thực hiện, chúng em đã làm việc hết mình để hoàn thành tốt nhiệm vụ được giao. Kết quả đạt được không chỉ là thành quả của sự học hỏi mà còn là động lực để chúng em tiếp tục chinh phục và phát triển thêm những kiến thức mới trong lĩnh vực này. Chúng em xin gửi lời cảm ơn chân thành đến giảng viên đã luôn tận tâm hỗ trợ và hướng dẫn, tạo điều kiện thuận lợi để chúng em hoàn thành bài tập lớn này.

Tài liệu tham khảo

[1] Art of Problem. (2012, July 31). *Public Key Cryptography: RSA Encryption Algorithm.* [YouTube]. Retrieved from https://youtu.be/wXB-V_Keiu8?si=OVSGt8JsYtVH9Fem

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2001). *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. ISBN 0262032937. Section 31.7: The RSA public-key cryptosystem, pp. 881–887.

[3] Miller, G. L. (1976). *Riemann's Hypothesis and Tests for Primality.* Journal of Computer and System Sciences, 13(3), 300–317. 10.1145/800116.803773, S2CID 10690396.

[4] Rabin, M. O. (1980). *Probabilistic algorithm for testing primality.* Journal of Number Theory, 12(1), 128–138. 10.1016/0022-314X(80)90084-0.

[5] Tài liệu học thuật. *Mật Mã và An Ninh Mạng - Chương III - Mã hóa khóa công khai.*

Phụ lục: Giới thiệu thư viện NTL

NTL (*Number Theory Library*) là một thư viện C++ mạnh mẽ và phổ biến, hỗ trợ các phép toán trong lý thuyết số và đại số. Thư viện được thiết kế để xử lý các bài toán phức tạp với hiệu năng cao, đặc biệt hữu ích trong các lĩnh vực như mật mã học, toán học tính toán, và lý thuyết số.

Một số tính năng nổi bật của NTL:

- Hỗ trợ các phép toán trên số nguyên lớn (arbitrary precision integers).
- Cung cấp các công cụ để làm việc với đa thức, ma trận, và số học mô-đun.
- Tích hợp các thuật toán kiểm tra tính nguyên tố, bao gồm kiểm tra Miller-Rabin.
- Tối ưu hóa hiệu năng thông qua các thư viện như GMP (GNU Multiple Precision Arithmetic Library).
- Hỗ trợ mở rộng và tích hợp dễ dàng với các dự án mã nguồn mở khác.

NTL thường được sử dụng trong các ứng dụng nghiên cứu học thuật và công nghiệp, đặc biệt trong lĩnh vực mật mã học, nơi mà các bài toán liên quan đến số học lớn và tính toán mô-đun xuất hiện thường xuyên.

Tài liệu chi tiết và mã nguồn của NTL có sẵn tại:

1. Trang chính thức: https://www.shoup.net/ntl/
2. Kho lưu trữ mã nguồn: https://github.com/libntl/ntl