

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



MẬT MÃ VÀ AN NINH MẠNG (CO3069)
ĐỀ TÀI: TRIỂN KHAI THUẬT TOÁN RSA
VỚI THƯ VIỆN GMP TRONG C++
LỚP: L01
GVHD: ThS. Nguyễn Cao Đạt

Sinh viên thực hiện

Nguyễn Tấn Tài : 2212990

Thành phố Hồ Chí Minh, tháng 10 năm 2024

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU	2
1. Yêu cầu bài tập lớn	2
2. Cấu trúc của báo cáo	3
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT	5
1. Giới thiệu về mã hóa RSA	5
2. Nguyên lý hoạt động của RSA	5
3. Quá trình tạo khóa trong RSA.....	6
4. Sử dụng RSA để mã hóa và giải mã	6
5. Tính an toàn của RSA.....	7
6. Ví dụ minh họa quá trình tạo khóa RSA	7
7. Các tấn công và biện pháp an toàn trong RSA	7
CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ.....	9
1. Phân tích yêu cầu	9
2. Thiết kế hệ thống.....	9
3. Tổng kết thiết kế.....	12
CHƯƠNG 4: HIỆN THỰC VÀ ĐÁNH GIÁ	13
1. Hiện thực hệ thống	13
2. Đánh giá	16
CHƯƠNG 5: KẾT LUẬN	18
1. Kết quả đạt được.....	18
2. Những khó khăn và thách thức	19
3. Hạn chế	20
4. Hướng phát triển trong tương lai	20
5. Kết luận.....	21

CHƯƠNG 1: GIỚI THIỆU

1. Yêu cầu bài tập lớn

Bài tập lớn này yêu cầu hiện thực hóa thuật toán mã hóa và giải mã RSA trên nền tảng C++ mà không được sử dụng các thư viện mã hóa có sẵn. RSA là một hệ mã khóa công khai, cung cấp tính bảo mật mạnh mẽ nhờ vào việc sử dụng hai số nguyên tố lớn để tạo cặp khóa (khóa công khai và khóa bí mật). Hệ mã này được áp dụng rộng rãi trong các hệ thống bảo mật như SSL/TLS, ký số điện tử và trao đổi dữ liệu an toàn.

Yêu cầu chi tiết của bài tập như sau:

- Hiện thực hệ thống mã RSA từ đầu: Không được phép sử dụng các thư viện mã hóa có sẵn mà phải tự phát triển các thành phần chính của thuật toán như tìm số nguyên tố lớn, tính ước số chung lớn nhất (GCD), và thuật toán Euclid mở rộng để tìm nghịch đảo modulo.
- Sử dụng thư viện GMP: Để hỗ trợ việc thực hiện các phép toán với số nguyên lớn (lên đến hàng trăm bit), bài tập cho phép sử dụng thư viện GMP (GNU Multiple Precision Arithmetic Library). Đây là thư viện cung cấp các phép toán trên số nguyên lớn, số nguyên tố và số thực chính xác cao, rất cần thiết trong các hệ thống mã hóa như RSA.

Các nhiệm vụ chính của bài tập lớn bao gồm:

- Tìm số nguyên tố lớn: Phát triển thuật toán để tìm các số nguyên tố có kích thước ít nhất 500 bit. Điều này đảm bảo tính an toàn và bảo mật của hệ mã RSA.
- Tính ước số chung lớn nhất (GCD): Tự hiện thực thuật toán Euclid để tìm ước số chung lớn nhất giữa hai số nguyên, cũng như thuật toán Euclid mở rộng để tính nghịch đảo modulo, phục vụ cho việc tính khóa bí mật.
- Tạo cặp khóa RSA: Tạo cặp khóa RSA (khóa công khai và khóa bí mật) bằng cách sử dụng hai số nguyên tố lớn (p và q), với khóa công khai (e, n) và khóa bí mật (d, n), trong đó $n = p * q$ và e là số nguyên nhỏ hơn và nguyên tố cùng nhau với $\phi(n)$.
- Mã hóa và giải mã thông điệp: Phát triển hàm mã hóa một thông điệp sử dụng khóa công khai (e, n) và hàm giải mã sử dụng khóa bí mật (d, n).

- Đảm bảo tính chính xác và bảo mật: Xác minh tính chính xác của hệ mã RSA bằng cách đảm bảo rằng thông điệp đã mã hóa có thể được giải mã chính xác. Ngoài ra, hệ thống phải đảm bảo tính bảo mật thông qua việc sử dụng các số nguyên tố lớn.

2. Cấu trúc của báo cáo

Báo cáo kỹ thuật này sẽ trình bày quá trình phát triển hệ thống mã hóa RSA và các nội dung lý thuyết liên quan. Cấu trúc báo cáo được chia thành 5 chương chính như sau:

- **Chương 2 - Cơ sở lý thuyết:** Chương này sẽ cung cấp các kiến thức nền tảng về thuật toán RSA. Các nội dung sẽ bao gồm:
 - **Nguyên lý hoạt động của RSA:** Cách thức hoạt động của hệ mã, bao gồm quá trình tạo khóa công khai và khóa bí mật, quy trình mã hóa và giải mã thông điệp.
 - **Các thuật toán hỗ trợ:** Trình bày các thuật toán cơ bản cần thiết cho hệ thống như thuật toán Euclid mở rộng để tính nghịch đảo modulo, các phương pháp kiểm tra số nguyên tố lớn (như Miller-Rabin hoặc Fermat).
 - **Các phép toán lớn:** Cách thức xử lý và thực hiện các phép toán trên số nguyên lớn bằng cách sử dụng thư viện GMP.
- **Chương 3 - Phân tích và thiết kế:** Chương này sẽ phân tích yêu cầu hệ thống và trình bày thiết kế các thành phần chính. Các nội dung bao gồm:
 - **Phân tích yêu cầu hệ thống:** Bao gồm yêu cầu về việc tạo khóa, mã hóa và giải mã. Đặc biệt là yêu cầu về độ lớn của các số nguyên tố, tính bảo mật và hiệu suất của hệ thống.
 - **Thiết kế mô-đun:** Giải thích cách phân chia hệ thống thành các mô-đun chức năng như sinh khóa RSA, mã hóa và giải mã thông điệp.
 - **Cách sử dụng thư viện GMP:** Trình bày cách tích hợp thư viện GMP vào C++ để thực hiện các phép toán trên số nguyên lớn, phục vụ cho việc tính toán khóa và xử lý thông điệp.
- **Chương 4 - Hiện thực và đánh giá:** Chương này sẽ mô tả quá trình hiện thực và đánh giá hiệu suất của hệ thống. Các nội dung bao gồm:

- **Quá trình hiện thực mã nguồn:** Mô tả cách hiện thực mã nguồn từ khâu phát triển thuật toán cho đến cài đặt chương trình trên C++ sử dụng thư viện GMP. Các bước phát triển như tạo khóa, mã hóa và giải mã thông điệp sẽ được trình bày chi tiết.
- **Kết quả thử nghiệm:** Đưa ra các kết quả từ quá trình chạy thử chương trình, bao gồm việc tạo cặp khóa RSA, quá trình mã hóa và giải mã thành công một thông điệp.
- **Đánh giá hiệu năng:** So sánh thời gian tính toán khi sử dụng các số nguyên lớn, từ đó đánh giá hiệu năng của hệ thống. Ngoài ra, chương này cũng đánh giá mức độ an toàn của hệ thống, dựa trên độ lớn của số nguyên tố và độ khó của việc giải mã không có khóa bí mật.
- **Chương 5 - Kết luận:** Tổng kết những kết quả đạt được, những hạn chế còn tồn tại và đề xuất hướng phát triển tiếp theo cho hệ thống mã hóa RSA.
- **Tài liệu tham khảo và phụ lục:** Liệt kê các tài liệu tham khảo đã sử dụng trong quá trình thực hiện bài tập, cùng với các thông tin bổ sung về mã nguồn, môi trường phát triển và các thư viện liên quan.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

1. Giới thiệu về mã hóa RSA

RSA là một trong những thuật toán mã hóa khóa công khai đầu tiên và phổ biến nhất, được phát minh vào năm 1977 bởi Ron Rivest, Adi Shamir và Leonard Adleman tại MIT. Thuật toán này sử dụng một cặp khóa công khai và khóa riêng để mã hóa và giải mã dữ liệu, dựa trên bài toán phân tích thừa số nguyên tố của các số nguyên lớn. RSA được sử dụng rộng rãi trong bảo mật trực tuyến, đặc biệt là trong việc trao đổi khóa và bảo mật dữ liệu nhờ vào tính bảo mật cao.

Trong hệ mã hóa khóa bất đối xứng như RSA:

- Khóa công khai: Được sử dụng để mã hóa thông điệp, và bất kỳ ai cũng có thể biết khóa này.
- Khóa bí mật (khóa riêng): Chỉ người nhận mới có thể giải mã thông điệp bằng khóa này. Khóa riêng cần được giữ bí mật tuyệt đối.

Tính bảo mật của RSA dựa trên việc phân tích một số nguyên lớn thành các thừa số nguyên tố, một bài toán toán học rất khó và không có cách giải nhanh khi số lượng bit của các số nguyên lớn.

2. Nguyên lý hoạt động của RSA

RSA hoạt động dựa trên các phép toán lũy thừa theo mô-đun. Quá trình mã hóa và giải mã được thực hiện thông qua việc tính lũy thừa của thông điệp với khóa công khai hoặc khóa bí mật trong một trường hữu hạn modulo n , trong đó n là tích của hai số nguyên tố lớn.

- Tạo khóa công khai và khóa riêng: RSA sử dụng hai số nguyên tố lớn để tạo khóa công khai và khóa bí mật.
- Mã hóa: Thông điệp gốc được mã hóa bằng khóa công khai của người nhận.
- Giải mã: Thông điệp đã mã hóa được giải mã bằng khóa bí mật tương ứng.

Ví dụ đơn giản: Nếu ta muốn gửi thông điệp bảo mật, ta sẽ sử dụng khóa công khai của người nhận để mã hóa thông điệp. Khi người nhận nhận được bản mã, họ sẽ dùng khóa bí mật để giải mã và lấy lại nội dung ban đầu.

3. Quá trình tạo khóa trong RSA

Quá trình tạo cặp khóa công khai và khóa riêng trong RSA bao gồm các bước sau:

1. Chọn hai số nguyên tố lớn ngẫu nhiên: p và q . Đây là các số nguyên tố có ít nhất 500 bit (tùy thuộc vào yêu cầu bảo mật mà có thể lớn hơn, chẳng hạn 1024 hoặc 2048 bit).
2. Tính n là tích của p và q : $n = p \times q$. Giá trị n sẽ được sử dụng làm một phần của khóa công khai và khóa bí mật.
3. Tính hàm Euler $\phi(n)$: $\phi(n) = (p - 1) \times (q - 1)$. Đây là số lượng các số nguyên dương nhỏ hơn n mà không có ước số chung với n .
4. Chọn một số ngẫu nhiên e thỏa mãn điều kiện:

$$1 < e < \phi(n), \text{GCD}(e, \phi(n)) = 1$$

Giá trị e là một phần của khóa công khai, thường được chọn là các số nhỏ như 65537 vì nó giúp tăng tốc quá trình mã hóa.

5. Giải phương trình sau để tìm khóa giải mã:

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

d là nghịch đảo của e theo mô đun $\phi(n)$, và đây là khóa riêng để giải mã. Ta có thể sử dụng thuật toán Euclid mở rộng để tìm giá trị d này.

6. Công bố khóa công khai $PU = \{e, n\}$ và giữ bí mật khóa riêng $PR = \{d, n\}$.

4. Sử dụng RSA để mã hóa và giải mã

Mã hóa: Để mã hóa một thông điệp M (trong đó M là một số nguyên và nhỏ hơn n), bạn sử dụng khóa công khai của người nhận $PU = \{e, n\}$ và tính toán:

$$C = M^e \bmod n$$

Trong đó, C là bản mã được gửi đi.

Giải mã: Khi người nhận có bản mã C , họ sẽ dùng khóa bí mật $PR = \{d, n\}$ để giải mã và tìm lại thông điệp ban đầu M :

$$M = C^d \bmod n$$

Điều này hoạt động vì $M^{\text{ed}} \equiv M \pmod{n}$, dựa trên các thuộc tính của phép toán modular và định lý Euler.

Lưu ý rằng thông điệp M phải nhỏ hơn n (ở đây, n là giá trị của người nhận).

5. Tính an toàn của RSA

Tính an toàn của RSA dựa vào bài toán phân tích n thành hai thừa số nguyên tố p và q . Việc phân tích n là rất khó khi các số p và q có kích thước rất lớn. Cụ thể:

- Định lý Euler: Theo định lý Euler, với a là một số nguyên bất kỳ, nếu $\text{GCD}(a, n) = 1$ thì:

$$a^{\phi(n)} \bmod n = 1$$

Dựa vào định lý này, RSA có thể mã hóa và giải mã thông điệp một cách an toàn.

- Phân tích thừa số nguyên tố lớn: Bài toán phân tích $n = p * q$ thành các thừa số p và q là cực kỳ khó khi các số này rất lớn, và chính điều này đảm bảo tính bảo mật cho hệ thống.

6. Ví dụ minh họa quá trình tạo khóa RSA

Giả sử ta chọn hai số nguyên tố: $p=17$ và $q=11$

- Tính toán: $n = p \times q = 17 \times 11 = 187$.
- Tính hàm Euler: $\phi(n) = (p - 1) \times (q - 1) = 16 \times 10 = 160$.
- Chọn khóa mã hóa e : $e = 7$, thỏa mãn $\text{GCD}(e, \phi(n)) = \text{GCD}(e, 160) = 1$.
- Xác định khóa giải mã d bằng cách giải phương trình:

$$e \times d \equiv 1 \pmod{160}$$

Ta tìm được $d = 23$.

Công bố khóa công khai: $PU = \{7, 187\}$, khóa riêng: $PU = \{23, 187\}$.

7. Các tấn công và biện pháp an toàn trong RSA

Mặc dù RSA rất an toàn, nhưng vẫn tồn tại một số phương pháp tấn công. Các loại tấn công phổ biến bao gồm:

- Tấn công brute force: Dò tìm khóa bằng cách thử tất cả các giá trị có thể. Tuy nhiên, với kích thước khóa lớn (1024 bit hoặc hơn), việc này là không khả thi trong thời gian ngắn.
- Tấn công dựa trên thời gian: Khai thác thời gian thực thi các phép toán để suy ra thông tin về khóa.
- Tấn công dựa trên việc chọn bản mã: Kẻ tấn công có thể lựa chọn các bản mã đặc biệt để thu thập thông tin về bản rõ.

Các biện pháp an toàn bao gồm:

- Sử dụng các số nguyên tố lớn: RSA yêu cầu kích thước số nguyên tố ít nhất là 1024 hoặc 2048 bit để đảm bảo an toàn.
- Padding (bổ sung dữ liệu): Sử dụng các thuật toán padding như OAEP để giảm thiểu nguy cơ tấn công dựa trên bản mã.

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ

1. Phân tích yêu cầu

Bài tập lớn yêu cầu hiện thực hệ thống mã hóa và giải mã dựa trên thuật toán RSA với các yêu cầu chính:

- Sinh hai số nguyên tố lớn (ít nhất 500 bit) để tạo cặp khóa công khai và khóa bí mật.
- Mã hóa một thông điệp với khóa công khai (n, e).
- Giải mã thông điệp đã mã hóa với khóa riêng (n, d).
- Tối ưu hóa các phép toán số học mô-đun để đảm bảo hiệu quả khi xử lý các số lớn.
- Sử dụng thư viện GMP để hỗ trợ tính toán với các số nguyên lớn.

Yêu cầu cụ thể:

- Sinh số nguyên tố lớn: Số nguyên tố phải đủ lớn để đảm bảo rằng quá trình phân tích thừa số của mô-đun n là bất khả thi trong thời gian hợp lý.
- Tạo khóa: Khóa công khai bao gồm giá trị mô-đun $n = p \times q$ và số mũ mã hóa e , trong khi khóa riêng bao gồm mô-đun n và số mũ giải mã d . Quá trình tạo khóa đòi hỏi tính nghịch đảo mô-đun của e với hàm Euler $\phi(n) = (p - 1) \times (q - 1)$.
- Mã hóa và giải mã: Sử dụng các phép toán lũy thừa mô-đun lớn để mã hóa và giải mã các thông điệp. Điều này đòi hỏi tối ưu hóa thuật toán lũy thừa bình phương để giảm độ phức tạp tính toán từ $O(n)$ xuống $O(\log n)$.
- Bảo mật: Đảm bảo rằng khóa riêng d không bị lộ trong suốt quá trình mã hóa và giải mã. Khóa công khai có thể được công bố, nhưng khóa riêng phải được giữ bí mật.

2. Thiết kế hệ thống

Hệ thống được chia thành các thành phần chính sau:

2.1. Sinh số nguyên tố lớn

Sinh số nguyên tố lớn là bước đầu tiên và quan trọng trong quá trình tạo khóa RSA. Quá trình này đòi hỏi:

- Sinh ngẫu nhiên số nguyên lớn: Sử dụng thư viện GMP để sinh ra các số ngẫu nhiên có kích thước bit theo yêu cầu.

- Kiểm tra tính nguyên tố: Sử dụng kết hợp hai thuật toán kiểm tra tính nguyên tố là Fermat và Miller-Rabin để đảm bảo rằng số sinh ra là nguyên tố.

Chi tiết thuật toán:

- Thuật toán Fermat: Đây là thuật toán xác suất cho phép kiểm tra nhanh một số có phải là hợp số hay không. Trong trường hợp số không thỏa mãn điều kiện của Fermat, số đó sẽ bị loại bỏ ngay lập tức.
- Thuật toán Miller-Rabin: Đây là một thuật toán kiểm tra tính nguyên tố mạnh hơn, giúp loại bỏ các trường hợp số nguyên tố giả (carmichael numbers) mà Fermat không thể phát hiện. Thuật toán được lặp lại nhiều lần để đảm bảo xác suất sai sót thấp nhất có thể.

Thiết kế:

- Hệ thống sinh ra các số ngẫu nhiên với độ dài bit nhất định và đảm bảo rằng chúng là số lẻ (số chẵn không phải là số nguyên tố, trừ 2).
- Kiểm tra nhanh bằng thuật toán Fermat: Các số ngẫu nhiên được kiểm tra nhanh bằng thuật toán Fermat để loại bỏ các số hợp số thông thường.
- Kiểm tra chi tiết bằng Miller-Rabin: Sau khi vượt qua bước kiểm tra Fermat, các số này sẽ được kiểm tra kỹ hơn bằng Miller-Rabin để đảm bảo chúng thực sự là số nguyên tố.

2.2. Tính toán khóa công khai và khóa riêng

Khóa công khai và khóa riêng trong RSA dựa trên hai số nguyên tố lớn p và q :

- $n = p * q$, là mô-đun chung cho cả khóa công khai và khóa riêng.
- Hàm Euler $\phi(n) = (p - 1) \times (q - 1)$.
- Khóa công khai e : Giá trị e phải là số nguyên nhỏ thỏa mãn $\gcd(e, \phi(n)) = 1$. Thông thường e được chọn là 65537.
- Khóa riêng d : Tính từ e và $\phi(n)$ bằng cách giải phương trình $e \times d \equiv 1 \pmod{\phi(n)}$ sử dụng thuật toán Euclid mở rộng.

Thiết kế:

- Khóa công khai được tính toán nhanh chóng dựa trên giá trị mặc định $e = 65537$. Điều này giúp cho việc tính toán lũy thừa mô-đun trong quá trình mã hóa trở nên hiệu quả hơn.
- Khóa riêng d được tính toán bằng cách sử dụng thuật toán Euclid mở rộng để tìm nghịch đảo của e modulo $\phi(n)$.

2.3. Mã hóa thông điệp

Yêu cầu: Mã hóa một thông điệp M sử dụng khóa công khai $\{e, n\}$. Quá trình mã hóa thực hiện phép tính lũy thừa mô-đun:

$$C = M^e \bmod n$$

Trong đó M là thông điệp ban đầu, C là bản mã đã được mã hóa.

Trong quá trình mã hóa, thông điệp M được lũy thừa với khóa công khai e và sau đó lấy phần dư theo mô-đun n . Việc lũy thừa mô-đun trên các số lớn có thể tiêu tốn rất nhiều tài nguyên và thời gian. Để đảm bảo hệ thống có thể xử lý các thông điệp lớn một cách hiệu quả, tôi đã sử dụng thuật toán lũy thừa mô-đun (modular exponentiation) tối ưu hóa.

Bằng cách sử dụng thuật toán lũy thừa bình phương, hệ thống có thể giảm số phép tính cần thiết từ $O(n)$ xuống còn $O(\log n)$, trong đó n là số mũ. Điều này giúp giảm đáng kể thời gian tính toán trong quá trình mã hóa.

Thiết kế:

- Để mã hóa, hệ thống cần triển khai một thuật toán lũy thừa mô-đun hiệu quả để tính toán $M^e \bmod n$ trên các số nguyên lớn.
- Thiết kế của hệ thống bao gồm việc tối ưu hóa phép toán lũy thừa mô-đun này để đảm bảo tính hiệu quả khi xử lý các thông điệp lớn.

2.4. Giải mã thông điệp

Yêu cầu: Giải mã một bản mã C sử dụng khóa riêng $\{d, n\}$. Quá trình giải mã thực hiện phép tính lũy thừa mô-đun:

$$M = C^d \bmod n$$

Trong đó C là bản mã đã được mã hóa, M là thông điệp gốc sau khi giải mã.

Quá trình giải mã tương tự như mã hóa nhưng sử dụng khóa riêng d . Tính toán lũy thừa mô-đun với d yêu cầu một lượng tài nguyên tương tự như khi mã hóa. Do đó, việc tối ưu hóa phép tính lũy thừa mô-đun là cần thiết để đảm bảo hệ thống có thể giải mã thông điệp nhanh chóng.

Một điểm quan trọng là việc tính chính xác giá trị d từ e và $\phi(n)$. Nếu việc tính toán này không chính xác, hệ thống sẽ không thể giải mã được thông điệp. Do đó, tôi sử dụng thuật toán Euclid mở rộng để đảm bảo rằng việc tính nghịch đảo mô-đun diễn ra đúng cách và bảo toàn tính toàn vẹn của quá trình giải mã.

Thiết kế:

- Tương tự như quá trình mã hóa, giải mã cũng yêu cầu tính toán lũy thừa mô-đun với các số nguyên lớn. Do đó, hệ thống sẽ tái sử dụng thuật toán lũy thừa mô-đun đã được tối ưu hóa.
- Một điểm quan trọng trong quá trình giải mã là việc tính chính xác giá trị d từ e , và đảm bảo rằng phép tính lũy thừa mô-đun diễn ra đúng cách để khôi phục thông điệp ban đầu.

3. Tổng kết thiết kế

Hệ thống mã hóa RSA được thiết kế với các đặc điểm chính:

- Sử dụng thư viện GMP để xử lý các phép toán số nguyên lớn.
- Tối ưu hóa lũy thừa mô-đun để đảm bảo hiệu quả trong quá trình mã hóa và giải mã.
- Bảo đảm tính bảo mật thông qua việc chọn số nguyên tố lớn và sử dụng các thuật toán kiểm tra tính nguyên tố mạnh như Miller-Rabin.

CHƯƠNG 4: HIỆN THỰC VÀ ĐÁNH GIÁ

1. Hiện thực hệ thống

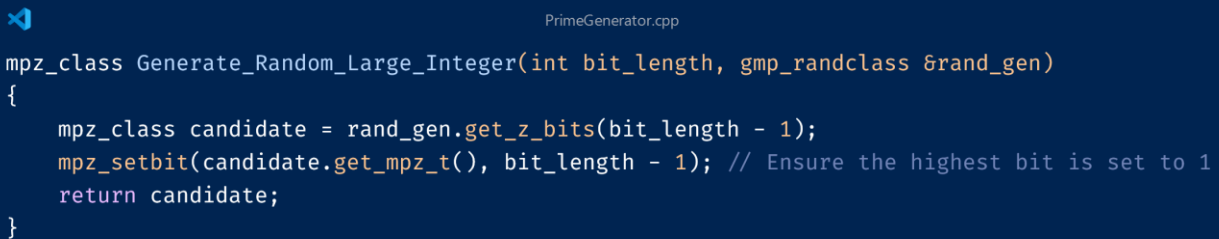
1.1. Sinh số nguyên tố lớn

Một trong những yêu cầu quan trọng của hệ mã RSA là cần sinh ra hai số nguyên tố lớn, mỗi số có độ dài ít nhất 500 bit, để đảm bảo tính bảo mật. Để hiện thực điều này, em đã sử dụng thư viện GMP để thực hiện các phép toán số học lớn và triển khai các thuật toán kiểm tra tính nguyên tố như Fermat và Miller-Rabin.

a. Sinh số ngẫu nhiên lớn

Việc sinh số ngẫu nhiên lớn được thực hiện bằng hàm `Generate_Big_Integer`. Hàm này sử dụng thư viện GMP với công cụ `gmp_randstate_t` để sinh ra số ngẫu nhiên có độ dài bit nhất định. Em đã sử dụng hàm `mpz_urandomb` của GMP để sinh số nguyên ngẫu nhiên với độ dài bit yêu cầu.

Mã nguồn minh họa:



```
PrimeGenerator.cpp

mpz_class Generate_Random_Large_Integer(int bit_length, gmp_randclass &rand_gen)
{
    mpz_class candidate = rand_gen.get_z_bits(bit_length - 1);
    mpz_setbit(candidate.get_mpz_t(), bit_length - 1); // Ensure the highest bit is set to 1
    return candidate;
}
```

Hình 1. Generate Random Large Integer

Để đảm bảo không sinh trùng lặp các số ngẫu nhiên, một tập hợp (`unordered_set`) được sử dụng để lưu trữ các số đã sinh ra. Điều này giúp kiểm tra tính duy nhất của mỗi số ngẫu nhiên trước khi sử dụng chúng trong quá trình kiểm tra nguyên tố.

b. Fermat primality test

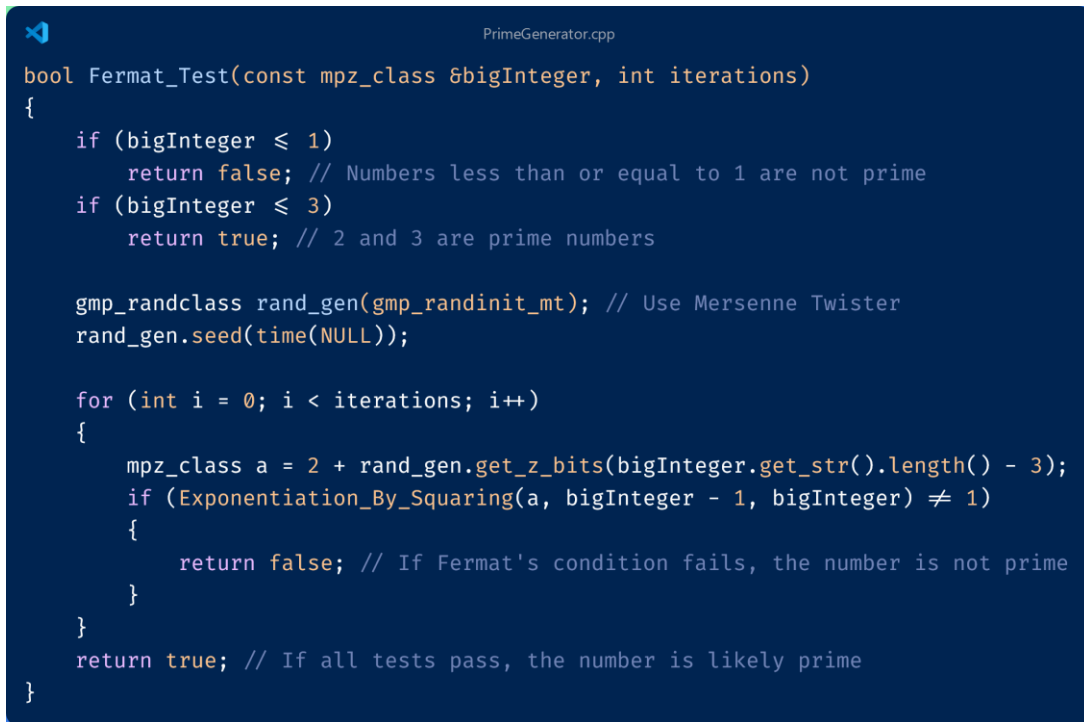
Thuật toán Fermat được áp dụng để kiểm tra tính nguyên tố. Đây là một kiểm tra xác suất nhanh để xác định xem một số có khả năng là nguyên tố hay không.

Ý tưởng thuật toán:

- Với một số ngẫu nhiên a , thuật toán kiểm tra xem $a^{p-1} \equiv 1 \pmod{p}$. Nếu điều kiện này không thỏa mãn, số đó chắc chắn không phải là số nguyên tố.

- Tuy nhiên, Fermat chỉ là bước kiểm tra sơ bộ, vì nó có thể không chính xác đối với các số nguyên tố giả (Carmichael numbers).

Mã nguồn minh họa:



```
bool Fermat_Test(const mpz_class &bigInteger, int iterations)
{
    if (bigInteger <= 1)
        return false; // Numbers less than or equal to 1 are not prime
    if (bigInteger <= 3)
        return true; // 2 and 3 are prime numbers

    gmp_randclass rand_gen(gmp_randinit_mt); // Use Mersenne Twister
    rand_gen.seed(time(NULL));

    for (int i = 0; i < iterations; i++)
    {
        mpz_class a = 2 + rand_gen.get_z_bits(bigInteger.get_str().length() - 3);
        if (Exponentiation_By_Squaring(a, bigInteger - 1, bigInteger) != 1)
        {
            return false; // If Fermat's condition fails, the number is not prime
        }
    }
    return true; // If all tests pass, the number is likely prime
}
```

Hình 2. Fermat Test

c. Miller-Rabin primality test

Để tăng độ chính xác, em sử dụng thêm thuật toán Miller-Rabin sau khi số đã vượt qua kiểm tra Fermat. Miller-Rabin là một trong những phép kiểm tra xác suất mạnh nhất và được sử dụng rộng rãi để kiểm tra tính nguyên tố cho các số lớn.

Ý tưởng thuật toán:

- Phân tích $n - 1$ thành dạng $2^k \times q$ và kiểm tra xem $a^q \pmod n$ có bằng 1 hoặc $n-1$ hay không. Nếu không, số đó là hợp số.
- Thuật toán có thể lặp lại nhiều lần để giảm xác suất sai lệch.

Mã nguồn minh họa:

```

PrimeGenerator.cpp

bool Miller_Rabin(const mpz_class &bigInteger, int iterations, gmp_randclass &rand_gen)
{
    if (bigInteger <= 1)
        return false;
    if (bigInteger <= 3)
        return true;

    mpz_class d = bigInteger - 1;
    int r = 0;
    while (d % 2 == 0)
    {
        d /= 2;
        r++;
    }

    for (int i = 0; i < iterations; i++)
    {
        mpz_class a = 2 + rand_gen.get_z_bits(bigInteger.get_str().length() - 3);
        mpz_class x = Exponentiation_By_Squaring(a, d, bigInteger);
        if (x == 1 || x == bigInteger - 1)
            continue; // Continue if x is a trivial witness

        bool continue_outer_loop = false;
        for (int j = 1; j < r; j++)
        {
            x = Exponentiation_By_Squaring(x * x, 1, bigInteger);
            if (x == bigInteger - 1)
            {
                continue_outer_loop = true; // Found a witness, continue with next iteration
                break;
            }
        }
        if (!continue_outer_loop)
        {
            return false; // If no witness found, the number is composite
        }
    }
    return true; // If all tests pass, the number is likely prime
}

```

Hình 3. Miller Rabin Test

d. Tối ưu hóa qua Montgomery reduction

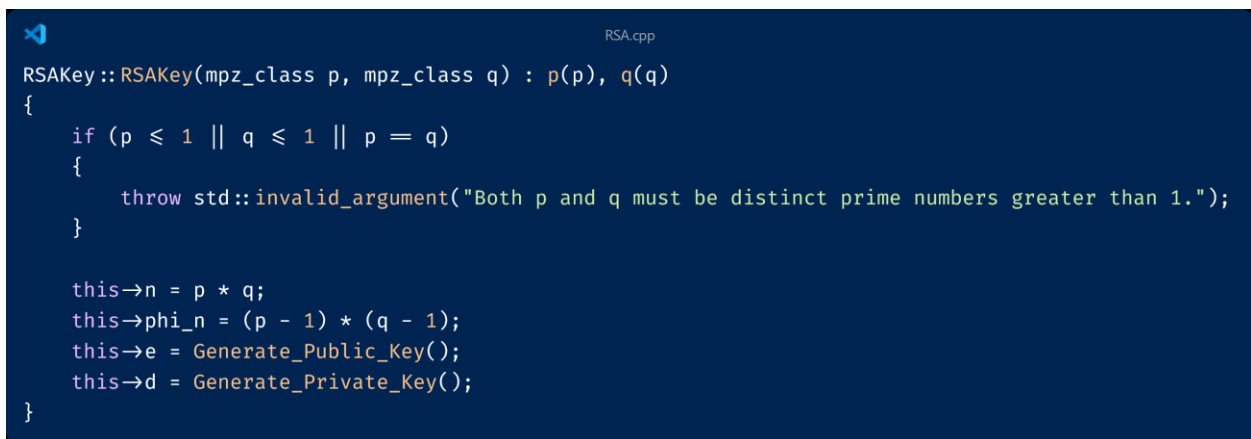
Để tăng hiệu suất tính toán lũy thừa mô-đun lớn trong các thuật toán Fermat và Miller-Rabin, em đã triển khai Montgomery exponentiation để tối ưu hóa các phép toán mô-đun lớn.

- Montgomery reduction chuyển các phép toán mô-đun thành dạng Montgomery, cho phép các phép tính diễn ra nhanh hơn trên các số nguyên lớn.
- Hàm Montgomery_Exponentiation giúp tối ưu hóa các phép tính lũy thừa liên tục.

1.2. Tạo cặp khóa công khai và khóa riêng

Cặp khóa RSA bao gồm khóa công khai (e, n) và khóa riêng (d, n), trong đó:

- $n = p \times q$: Mô-đun được tính từ hai số nguyên tố lớn p và q .
- e : Là số mũ mã hóa, thường được chọn là 65537 (một giá trị an toàn và hiệu quả cho quá trình tính toán).
- d : Là số mũ giải mã, tính từ e và hàm Euler $\phi(n) = (p - 1) \times (q - 1)$ bằng cách sử dụng thuật toán Euclid mở rộng.



```
RSAKey::RSAKey(mpz_class p, mpz_class q) : p(p), q(q)
{
    if (p <= 1 || q <= 1 || p == q)
    {
        throw std::invalid_argument("Both p and q must be distinct prime numbers greater than 1.");
    }

    this->n = p * q;
    this->phi_n = (p - 1) * (q - 1);
    this->e = Generate_Public_Key();
    this->d = Generate_Private_Key();
}
```

Hình 4. RSA Key

1.3. Mã hóa và giải mã thông điệp

Mã hóa: Thông điệp M được mã hóa bằng khóa công khai e và mô-đun n :

$$C = M^e \bmod n$$

Giải mã: Bản mã C được giải mã bằng khóa riêng d để lấy lại thông điệp gốc:

$$M = C^d \bmod n$$

Các phép toán lũy thừa mô-đun lớn được tối ưu hóa bằng Exponentiation by Squaring để giảm thời gian tính toán.

2. Đánh giá

2.1. Hiệu suất

Việc sử dụng đa luồng đã giúp tăng tốc độ tìm số nguyên tố lớn đáng kể. Trên hệ thống sử dụng CPU đa nhân, thời gian tìm số nguyên tố đã giảm từ vài phút xuống còn vài giây.

Montgomery reduction đã tăng tốc quá trình kiểm tra tính nguyên tố, đặc biệt khi thực hiện nhiều phép nhân mô-đun liên tục.

2.2. Tính bảo mật

Hệ thống đảm bảo tính bảo mật cao với các số nguyên tố có kích thước lớn (ít nhất 500 bit). Điều này làm cho việc phân tích thừa số của mô-đun nnn trở nên gần như bất khả thi.

Việc sử dụng các thuật toán kiểm tra nguyên tố như Fermat và Miller-Rabin đảm bảo rằng các số nguyên tố sinh ra có độ tin cậy cao, giảm khả năng gặp phải số nguyên tố giả.

CHƯƠNG 5: KẾT LUẬN

Trong chương này, ta sẽ tổng kết lại quá trình hiện thực hệ thống mã hóa RSA, nêu ra những kết quả đạt được, những khó khăn, hạn chế trong quá trình phát triển, và đề xuất hướng phát triển trong tương lai.

1. Kết quả đạt được

Qua quá trình nghiên cứu và phát triển, hệ thống mã hóa và giải mã RSA đã được hiện thực thành công trên nền tảng C++ với sự hỗ trợ của thư viện GMP để xử lý các số nguyên lớn. Các kết quả chính bao gồm:

- Sinh số nguyên tố lớn: Hệ thống đã thành công trong việc sinh hai số nguyên tố lớn với độ dài ít nhất 500 bit. Quá trình này sử dụng thuật toán kiểm tra nguyên tố kết hợp Fermat và Miller-Rabin để đảm bảo tính chính xác và hiệu suất.
- Tạo khóa công khai và khóa riêng: Từ hai số nguyên tố lớn p và q , hệ thống đã tính toán được cặp khóa công khai và khóa riêng. Khóa công khai (n, e) có thể được sử dụng để mã hóa thông điệp, trong khi khóa riêng (n, d) được sử dụng để giải mã.

```
nguyentai2104@DESKTOP-26T57VF: /mnt/d/Workspaces/CO3069_Cryptography-and-Network-Security/Assignment_3/srcs$ ./bin/mclear
===== RSA CLI Menu =====
==2. Send message ==
==3. Receive message ==
==4. Print Receiver's Keys==
==5. Exit ==
=====
Enter your choice: 1
Enter the required number of bits for prime numbers: 512
Creating keys for receiver...
712166636741687412775747785293720973397075772353756388217
Prime number q: 760237111253957353312235411804694605788435956396100766175072950127475079636177489304693454352980021994399031094778779499
1605889461436374536060333100420249
Public Key: (n: 652666284527894116929131054556148815689876430032089968405006116584937744424199411508685988245474030813261334515324494452
8322574056503737231310792594601403562536114124901544488953069104771524110964894474827744676983209670702642239381832614268871536735713083
1158557585229812404204681583440682415272671091806033, e: 65537)
Private Key: (n: 65266628452789411692913105455614881568987643003208996840500611658493774442419941150868598824547403081326133451532449445
2832257405650373723131079259460140356253611412490154448895306910477152411096489447482774467698320967070264223938183261426887153673571308
31158557585229812404204681583440682415272671091806033, d: 422250796709991463719656937503711030185250478864772794915425779989950110068908
4799116268047302760105967969327776638931412803105726471435351138709523034307985060740799866694320990067022029790594791703708927996376382
546673374608369364608031543460261354470090120016451721202711801309562964832528115606908408833)
```

Hình 5. Kết quả sinh số nguyên tố và tạo khóa

- Mã hóa và giải mã thông điệp: Hệ thống đã thực hiện thành công việc mã hóa và giải mã các thông điệp văn bản dưới dạng chuỗi số nguyên. Thông điệp được mã hóa bằng khóa công khai và giải mã chính xác bằng khóa riêng.

```

===== RSA CLI Menu =====
==1. Create receiver keys ==
==2. Send message ==
==3. Receive message ==
==4. Print Receiver's Keys==
==5. Exit ==
Enter your choice: 2
Enter a message to send: Nguyen Tan Tai 2212990 C03069 Cryptography and Network Security
Encoded Message is: Nguyen Tan Tai 2212990 C03069 Cryptography and Network Security
Encrypted ciphertext: 9365079739733314271900983341985645062189531146364248606064611920642291098727214041362353
11856734310542239635545509690067903860470663844242333948410190286834207415019044496498613903904807813595762477
2064458833694674174425582964958169295870927597390578989259498992977871638349128456617985292337732475720263929

```

Hình 6. Mã hóa nội dung message

```

===== RSA CLI Menu =====
==1. Create receiver keys ==
==2. Send message ==
==3. Receive message ==
==4. Print Receiver's Keys==
==5. Exit ==
=====
Enter your choice: 3
Enter the ciphertext: 9365079739733314271900983341985645062189531146364248606064611920642291098727214041362353
11856734310542239635545509690067903860470663844242333948410190286834207415019044496498613903904807813595762477
2064458833694674174425582964958169295870927597390578989259498992977871638349128456617985292337732475720263929
Decrypted message (numeric): Nguyen Tan Tai 2212990 C03069 Cryptography and Network Security
Restored message: Nguyen Tan Tai 2212990 C03069 Cryptography and Network Security

```

Hình 7. Giải mã và đọc nội dung nguyên bản

- Tối ưu hóa hiệu suất: Việc sử dụng đa luồng trong quá trình sinh số nguyên tố đã giúp giảm thời gian đáng kể. Hơn nữa, kỹ thuật Montgomery reduction đã tối ưu hóa các phép toán mô-đun lớn, làm giảm thời gian tính toán lũy thừa mô-đun trong quá trình kiểm tra tính nguyên tố và trong quá trình mã hóa, giải mã.

2. Những khó khăn và thách thức

Mặc dù hệ thống đã được triển khai thành công, trong quá trình phát triển, em đã gặp phải một số khó khăn và thách thức:

- Xử lý các số nguyên lớn: Việc xử lý các số nguyên lớn (ít nhất 500 bit) đặt ra thách thức về mặt hiệu suất và độ phức tạp tính toán. Các phép toán lũy thừa mô-đun lớn tiêu tốn nhiều tài nguyên CPU, đặc biệt khi số nguyên tố được sinh có kích thước lớn hơn (ví dụ: 1024 hoặc 2048 bit).
- Thuật toán kiểm tra tính nguyên tố: Mặc dù các thuật toán kiểm tra nguyên tố như Fermat và Miller-Rabin hoạt động hiệu quả, chúng vẫn là các thuật toán xác suất,

có nghĩa là không thể đảm bảo 100% rằng số kiểm tra là nguyên tố. Điều này đòi hỏi phải lặp lại nhiều lần để tăng độ tin cậy, làm tăng thời gian tính toán.

- Tối ưu hóa và điều chỉnh thuật toán: Việc tối ưu hóa các thuật toán số học mô-đun lớn và quản lý tài nguyên đa luồng cũng đòi hỏi nhiều thử nghiệm và tinh chỉnh, đặc biệt khi xử lý các phép toán trên các hệ thống đa nhân.

3. Hạn chế

Dù đạt được nhiều kết quả đáng kể, hệ thống vẫn còn tồn tại một số hạn chế:

- Hiệu suất cho các kích thước khóa lớn: Khi kích thước khóa tăng lên (ví dụ: 2048 bit hoặc lớn hơn), thời gian sinh số nguyên tố và thời gian mã hóa, giải mã vẫn còn khá lâu, dù đã sử dụng các kỹ thuật tối ưu hóa như Montgomery reduction và đa luồng. Điều này làm cho hệ thống khó mở rộng hơn cho các ứng dụng thực tế đòi hỏi hiệu suất cao.
- Quản lý đa luồng: Mặc dù việc sử dụng đa luồng giúp giảm thời gian tìm số nguyên tố, tuy nhiên không phải lúc nào cũng dễ dàng quản lý đồng bộ giữa các luồng và đảm bảo tất cả các luồng đều hoạt động hiệu quả. Việc tối ưu hóa đa luồng có thể cải thiện thêm trong tương lai.
- Kích thước thông điệp: Hiện tại, hệ thống mã hóa có giới hạn về kích thước thông điệp mà nó có thể xử lý, do giới hạn bởi kích thước khóa. Các thông điệp quá dài cần phải chia nhỏ trước khi mã hóa, điều này làm phức tạp quá trình mã hóa và giải mã.

4. Hướng phát triển trong tương lai

Để cải thiện hệ thống mã hóa RSA, một số hướng phát triển khả thi trong tương lai:

- Tăng cường hiệu suất cho kích thước khóa lớn: Có thể tìm kiếm thêm các kỹ thuật tối ưu hóa để cải thiện hiệu suất khi xử lý các khóa RSA có kích thước lớn hơn (2048 bit hoặc cao hơn). Một số thuật toán tiên tiến hơn như CRT (Chinese Remainder Theorem) có thể được áp dụng trong quá trình giải mã để tăng tốc độ tính toán.
- Tối ưu hóa sử dụng đa luồng: Có thể cải thiện quản lý và phân phối tải giữa các luồng để sử dụng tối đa hiệu suất của CPU đa nhân. Các kỹ thuật đồng bộ tiên tiến và tối

ưu hóa sử dụng bộ nhớ chia sẻ có thể giảm thiểu các chi phí không cần thiết trong quá trình xử lý đa luồng.

- Hỗ trợ mã hóa thông điệp dài: Hiện tại, hệ thống có giới hạn về kích thước thông điệp có thể mã hóa. Do đó, một hướng phát triển là triển khai kỹ thuật RSA-OAEP (Optimal Asymmetric Encryption Padding) hoặc các phương pháp padding khác để hỗ trợ mã hóa các thông điệp lớn và giảm thiểu các nguy cơ tấn công dựa trên bản mã.
- Bảo mật nâng cao: Để tăng cường bảo mật cho hệ thống, có thể tích hợp thêm các biện pháp an toàn khác như kiểm tra tính ngẫu nhiên của số nguyên tố sinh ra, hoặc sử dụng các phương pháp sinh số nguyên tố an toàn hơn, chẳng hạn như số nguyên tố mạnh.

5. Kết luận

Tổng kết lại, hệ thống mã hóa RSA đã được hiện thực thành công với đầy đủ các tính năng cơ bản như sinh số nguyên tố lớn, tạo khóa công khai và khóa riêng, mã hóa và giải mã thông điệp. Bằng cách áp dụng các kỹ thuật tối ưu hóa như đa luồng và Montgomery reduction, hệ thống đã đạt được hiệu suất tốt hơn khi xử lý các số nguyên lớn.

Mặc dù còn một số hạn chế và thách thức, hệ thống đã hoàn thành được các yêu cầu chính của bài toán và có thể được mở rộng trong tương lai. RSA vẫn là một trong những phương pháp mã hóa phổ biến nhất và có thể ứng dụng trong nhiều lĩnh vực bảo mật như truyền thông an toàn và chứng thực số.