BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC NHA TRANG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC TẬP CƠ SỞ

Tên đề tài: CÀI ĐẶT THUẬT TOÁN MÃ HÓA RSA VÀ HÀM BĂM SHA-256 TRONG PYTHON

Giảng viên hướng dẫn: ThS. Cấn Thị Phượng

Sinh viên thực hiện: Võ Văn Thành

Mã số sinh viên: 64132201

Khánh Hòa - tháng 01/2025

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC NHA TRANG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC TẬP CƠ SỞ

Tên đề tài: CÀI ĐẶT THUẬT TOÁN MÃ HÓA RSA VÀ HÀM BĂM SHA-256 TRONG PYTHON

Giảng viên hướng dẫn: ThS. Cấn Thị Phượng

Sinh viên thực hiện: Võ Văn Thành

Mã số sinh viên: 64132201

Khánh Hòa - tháng 01/2025

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG

LỜI CẢM ƠN

Trong suốt thời gian thực hiện đề tài, em đã nhận được sự giúp đỡ của quý phòng ban Trường Đại học Nha Trang nói chung và Khoa Công nghệ thông tin nói riêng đã tạo điều kiện tốt nhất cho em được hoàn thành đề tài. Đặc biệt là sự hướng dẫn tận tình của cô Ths. Cấn Thị Phượng đã giúp em hoàn thành tốt đề tài. Qua đây, em xin gửi lời cảm ơn sâu sắc đến sự giúp đỡ này.

Em xin chân thành cảm ơn!

Khánh Hòa, ngày … thàng … năm …

Tác giả bài báo cáo

(Ký và ghi rõ họ tên)

Chương 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1 LÝ DO CHỌN ĐỀ TÀI

Trong xã hội hiện đại ngày nay, Internet là một thứ không thể thiếu đối với tất cả mọi người. Tuy nhiên, bên cạnh những lợi ích cũng như sự tiện lợi mà Internet mang lại cho cuộc sống chúng ta, thì cũng xuất hiện không ít những rủi ro. Trong số những rủi ro đấy thì việc dữ liệu bị đánh cắp vẫn luôn xảy ra với rất nhiều người. Mỗi ngày, có rất nhiều người bị những kẻ lạ mặt mạo danh công an hoặc các cơ quan nhà nước gọi tới thông báo bản thân đang bị vướng vào một vụ điều tra bí mật và yêu cầu nạn nhân phải chuyển tiền và cung cấp một số thông tin cá nhân nếu không thì sẽ bị xử phạt. Hầu hết các nạn nhân đều nghe theo dù đã được cảnh báo là do những kẻ lừa đảo này nắm gần như là toàn bộ thông tin cá nhân của những nạn nhân.

Trong bối cảnh đó, việc bảo mật dữ liệu đang là ưu tiên hàng đầu của tất cả mọi người. Trong số những cách bảo mật thì sự ra đời của các thuật toán mã hóa và hàm băm có thể xem như là một bước thay đổi lớn. Nhờ những thuật toán này, dữ liệu được bảo mật một cách tốt hơn, tránh bị những kẻ lạ mặt đánh cắp.

Chính vì vậy, em quyết định lựa chọn đề tài **"Cài đặt thuật toán mã hóa RSA và hàm băm SHA-256 trong Python"** bởi vì đây đều là những thuật toán có tính ứng dụng tương đối cao, được sử dụng rộng rãi trong rất nhiều lĩnh vực như truyền thông, ngân hàng hay thương mại điện tử,… Bên cạnh đó, việc lựa chọn đề tài này còn là một sự đáp ứng hiệu quả cho sự nguy hiểm của việc dữ liệu bị đánh cắp bởi vì dữ liệu của mỗi người rất quan trọng. Em tin rằng, việc lựa chọn đề tài không chỉ là sự hỗ trợ cho việc bảo mật dữ liệu mà còn cung cấp cơ sở, nền tảng lý thuyết vững chắc để góp phần phát triển kiến thức của bản thân, từ đó em có thể đóng góp những điều tích cực cho cộng đồng bảo mật thông tin.

1.2 MỤC TIÊU NGHIÊN CỨU

1.2.1 Mục tiêu đề tài

- Tìm hiểu lý thuyết về các thuật toán mã hóa và hàm băm.

- Nắm vững các khái niệm về an toàn và bảo mật thông tin, bao gồm mã hóa đối xứng, mã hóa bất đối xứng và hàm băm.
- Hiểu được thuật toán mã hóa bất đối xứng RSA và hàm băm SHA-256.
- Minh họa kết quả bằng giao diện cụ thể.

1.2.2 Đối tượng và phạm vi nghiên cứu

- Đối tượng nghiên cứu: Thuật toán mã hóa RSA, hàm băm SHA-256 và ngôn ngữ lập trình Python.
- Phạm vi nghiên cứu: Ứng dụng nghiên cứu vào thực tế.

1.2.3 Phương pháp nghiên cứu

- Phương pháp nghiên cứu tài liệu.
- Phương pháp phân tích và so sánh.
- Phương pháp thực hành.
- Phương pháp tổng hợp.

Chương 2. CƠ SỞ LÝ THUYẾT

2.1 MÃ HÓA BẤT ĐỐI XỨNG RSA

2.1.1 Giới thiệu

Giải thuật mã hóa RSA là một giải thuật mã hóa dữ liệu không đối xứng vô cùng nổi tiếng được công bố bởi các nhà khoa học Ron Rivest, Adi Shamir và Leonard Adleman vào năm 1977 tại học viện MIT và được đặt tên từ các chữ cái đầu tiên trong tên của họ. Sở dĩ giải thuật này không đối xứng là do nó hoạt động dựa trên 2 khóa khác nhau là ***khóa công khai - public key và khóa riêng - private key***. Như tên gọi của chúng, khóa công khai cho phép tất cả mọi người đều có thể sử dụng thì khóa riêng lại chỉ cho phép duy nhất một người sử dụng.

Một số ví dụ về việc ứng dụng RSA có thể kể đến như việc một máy khách truy cập vào một trình duyệt bất kỳ và gửi một khóa công khai cho máy chủ để yêu cầu dữ liệu trong hệ thống client - server. Khi đó, máy chủ sẽ mã hóa dữ liệu cần gửi dựa trên khóa riêng mà máy khách gửi tới. Khi đã mã hóa xong, máy chủ sẽ gửi dữ liệu đã mã hóa cho máy khách. Cuối cùng, khi đã nhận được dữ liệu từ máy chủ thì máy khách chỉ cần việc giải mã là xong. Chính bởi vì việc mã hóa này không đối xứng nên không một ai có thể giải mã được trừ khi có một bên thứ ba có được cái khóa công khai từ trình duyệt. Điều này sẽ giúp bảo mật dữ liệu một cách tốt hơn.

2.1.2 Nguyên lý hoạt động

2.1.2.1 Một số khái niệm về lý thuyết số

Trước khi bước vào nguyên lý hoạt động của RSA, ta cần phải có một số kiến thức về lý thuyết số, cụ thể là phép chia modulo.

a. Phép chia modulo

Phép chia modulo chính là phép chia lấy phần dư. Ví dụ như $3 \bmod 2 = 1$ hoặc $5 \bmod 3 = 2$. Một cách tổng quát hơn, ta có

$$a \bmod n = r \text{ với } a \geq 0, n > 0, 0 \leq r \leq n - 1$$

Nếu hai số a và b cùng có số dư trong phép chia cho n thì ta nói a và b đồng dư trong phép chia modulo cho n và kí hiệu là

$$a \equiv b \ (\mathrm{mod}\ n) \text{ hoặc } a \equiv b \bmod n$$

Nếu $a \equiv 0 \bmod n$ thì ta nói a chia hết cho n hay n là ước số của a.

b. Một số tính chất của phép chia modulo

Cho a, b, n là các số nguyên dương. Khi đó phép chia modulo có các tính chất sau

- Phép cộng

$$(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$$

- Phép trừ

$$(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$$

- Phép nhân

$$(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$$

c. Phần tử nghịch đảo của phép nhân modulo

Nếu hai số nguyên a và n nguyên tố cùng nhau thì tồn tại số nguyên w thỏa mãn

$$a \cdot w \equiv 1 \bmod n$$

Ta gọi w là phần tử nghịch đảo của a trong phép modulo cho n và kí hiệu là a^{-1}.

Ví dụ: Do $n = 7$ và $a = 5$ là hai số nguyên tố cùng nhau nên ta tìm được $a^{-1} = 3$ vì

$$5 \cdot 3 = 15 \equiv 1 \bmod 7$$

d. Định lý Fermat

Định lý: Nếu p là số nguyên tố và a là số nguyên không chia hết cho p thì

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

e. Phép lũy thừa modulo

Ta định nghĩa phép lũy thừa modulo để tính y từ các số nguyên a, x, n như sau

$$y = a^x = (a \cdot a \cdots a) \bmod n \quad \text{với x số a nhân với nhau}$$

f. Thuật toán Euclid

Thuật toán Euclid dùng để tìm ước số chung lớn nhất của hai số nguyên a và b. Ta kí hiệu ước số chung lớn nhất này là $\gcd(a, b)$. Thuật toán này dựa trên định lý sau

Định lý: Với mọi số nguyên $a \geq 0$ và $b > 0$ thì ta có

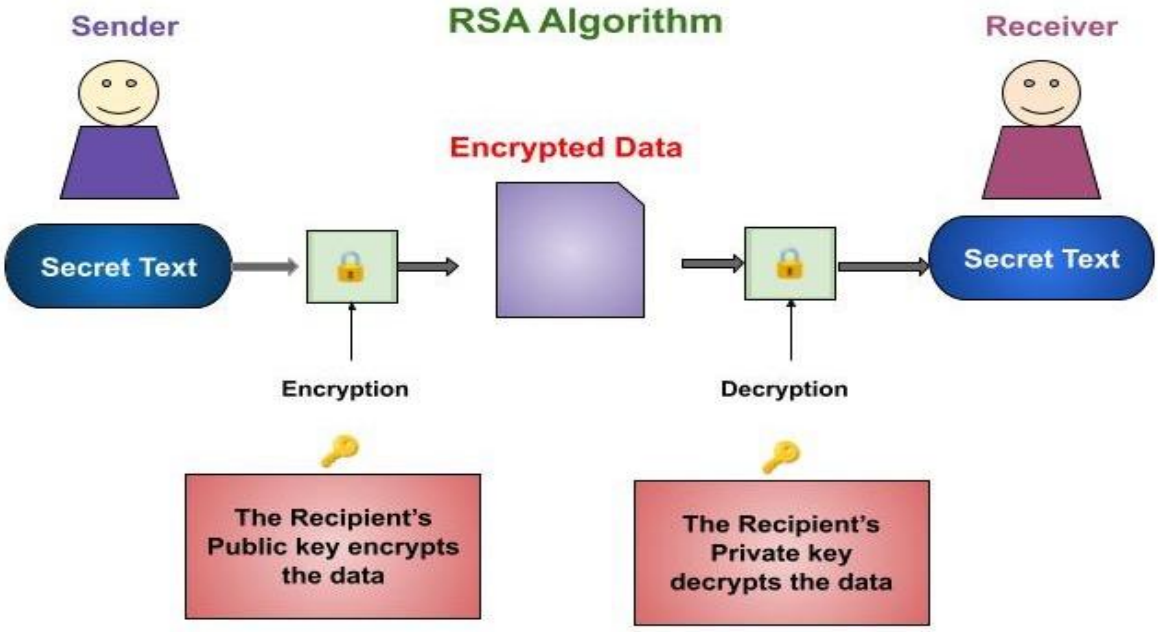
$$\gcd(a, b) = \gcd(b, a \bmod b)$$

Ngoài ra còn có thuật toán Euclid mở rộng. Đúng như tên gọi, thuật toán này mở rộng thuật toán Euclid ở chỗ trong trường hợp a và b nguyên tố cùng nhau, nghĩa là $\gcd(a, b) = 1$ với $a \geq 0, b > 0$. Khi đó thuật toán cho biết thêm giá trị nghịch đảo b^{-1} của b trong phép chia modulo a, hay

$$b \cdot b^{-1} \equiv 1 \bmod a$$

2.1.2.2 Nguyên tắc hoạt động của RSA

a. Ý tưởng



Hình 2-1. Mô tả cách mã hóa và giải mã của RSA

Về mặt tổng quát thì RSA là một phương pháp mã hóa theo khối, trong đó bản rõ M và bản mã C là các số nguyên từ 0 đến 2^i với i là số bits của khối và thường là 1024 bits. RSA sử dụng hàm một chiều là vấn đề phân tích một số thành thừa số nguyên tố. Và để thực hiện mã hóa và giải mã, RSA dùng phép lũy thừa modulo.

b. Các bước thực hiện

1. Chọn hai số nguyên tố lớn p, q và tính $N = pq$. Cần chọn p, q sao cho

$$M < 2^{i-1} < N < 2^i$$

Với $i = 1024$ thì N là một số nguyên dài khoảng 309 chữ số.

2. Tính $n = (p-1)(q-1)$.
3. Tìm một số nguyên e nguyên tố cùng nhau với n.
4. Tìm một số nguyên d là nghịch đảo của e trong phép modulo cho n.
5. Chọn khóa công khai $K_u = (e, N)$, khóa riêng $K_r = (d, N)$.
6. Việc mã hóa được thực hiện theo công thức
   - Phương án 1: Mã hóa bảo mật $C = E(M, K_u) = M^e \bmod N$.
   - Phương án 2: Mã hóa chứng thực $C = E(M, K_r) = M^d \bmod N$.
7. Việc giải mã được thực hiện theo công thức
   - Phương án 1: Mã hóa bảo mật $\overline{M} = D(C, K_r) = C^d \bmod N$.
   - Phương án 1: Mã hóa chứng thực $\overline{M} = D(C, K_u) = C^e \bmod N$.

Ví dụ về mã hóa RSA với kích thước khóa là 6 bits:

1. Chọn $p = 11, q = 3$ thì $N = pq = 33$ ($2^5 = 32 < 33 = 2^6$).

2. $n = (p-1)(q-1) = 20$.

3. Chọn $e = 3$ nguyên tố cùng nhau với n.

4. Tính nghịch đảo của e trong phép modulo cho n thì ta được $d = 7$.

5. Khóa công khai $K_u = (3,33)$, khóa riêng $K_r = (7,33)$.

Nếu thực hiện theo phương án 1:

6. Mã hóa bản rõ $M = 15$

$$C = M^e \bmod N = 15^3 \bmod 33 = 9$$

7. Giải mã bản mã $C = 9$

$$\overline{M} = C^d \bmod N = 9^7 \bmod 33 = 15 = M$$

Nếu thực hiện theo phương án 2:

6. Mã hóa bản rõ $M = 15$

$$C = M^d \bmod N = 15^7 \bmod 33 = 27$$

7. Giải mã bản mã $C = 9$

$$\overline{M} = C^e \bmod N = 27^3 \bmod 33 = 15 = M$$

2.1.3 Độ an toàn của RSA

Sau đây ta sẽ xem xét một số các tấn công phương pháp RSA

a. Vét cạn khóa: Cách tấn công này thủ tất cả các khóa có thể có để tìm ra bản giải mã có ý nghĩa, tương tự như cách thử khóa K của mã hóa đối xứng. Tất nhiên, với N đủ lớn thì việc tấn công là bất khả thi.

b. Phân tích N thành thừa số nguyên tố $N = pq$: Chúng ta đã nói rằng việc phân tích phải là bất khả thi thì mới là hàm một chiều. Đấy là nguyên tắc hoạt động của RSA. Tuy nhiên, nhiều thuật toán phân tích mới đã được đề xuất, cùng với tốc độ xử lý của máy tính ngày càng nhanh đã làm cho việc phân tích N thành thừa số nguyên tố không còn quá khó như trước. Năm 1977, các tác giả của RSA đã treo giải thưởng cho ai phá được RSA có kích thước của N vào khoảng 428 bits, tương đương 129 chữ số. Các tác giả này ước đoán phải mất 40 nghìn triệu năm mới có thể giải được. Tuy nhiên, vào năm 1994, câu đố này đã được giải chỉ trong vòng 8 tháng. Bảng sau đây liệt kê kích thước N của các RSA đã được phá mã cho đến hiện nay.

Số chữ số của N	*Số bits*	*Năm phá mã*	*Thuật toán*
100	332	1991	Quadratic sieve
110	365	1992	Quadratic sieve
120	398	1993	Quadratic sieve
129	428	1994	Quadratic sieve
130	431	1996	GNFS
140	465	1999	GNFS
155	512	1999	GNFS
160	530	2003	Lattice sieve
174	576	2003	Lattice sieve
200	633	2003	Lattice sieve
232	768	2009	GNFS

Bảng 2-1. Bảng liệt kê các mốc phá mã RSA

Tính đến hiện tại là năm 2025 thì thuật toán RSA với độ dài 1024 bits vẫn chưa bị phá. Tuy nhiên, với sự phát triển của máy tính

lượng tử thì các chuyên gia đều cho rằng việc bị phá hủy RSA với độ dài 1024 bits là hoàn toán có thể trong tương lai gần. Dĩ nhiên việc phá mã trên chỉ được thực hiện trong phòng thí nghiệm.

c. Đo thời gian: Đây là một phương pháp phá mã không dựa vào mặt toán học của thuật toán RSA, mà dựa vào một "hiệu ứng lề" sinh ra bởi quá trình giải mã RSA. Hiệu ứng lề đó là thời gian thực hiện giải mã. Giả sử người phá mã có thể đo được thời giải mã dùng thuật toán bình phương liên tiếp. Trong thuật toán bình phương liên tiếp, nếu một bits của d là 1 thì xảy ra hai phép modulo, nếu bits đó là 0 thì chỉ có một phép modulo, do đó thời gian thực hiện giải mã là khác nhau. Bằng một số phép thử chosen-plaintext, người phá mã có thể biết được các bits của d là 0 hay 1 và từ đó biết được d.

Phương pháp mã hóa này là một ví dụ cho thấy việc thiết kế một hệ mã an toàn rất phức tạp. Người thiết kế phải lường trước được hết các tình huống có thể xảy ra.

2.2 HÀM BĂM SHA-256

2.2.1 Giới thiệu

Hàm băm là một thuật toán nhận một chuỗi dữ liệu đầu vào và xuất ra một chuỗi kí tự với độ dài cố định và nhỏ. Chuỗi kí tự đầu ra đóng vai trò như "dấu vân tay" của dữ liệu đầu vào, nghĩa là một chuỗi dữ liệu đầu vào sẽ cho ra duy nhất một chuỗi kí tự đầu ra. Tuy nhiên, hàm băm là một chiều nên không thể suy được dữ liệu ban đầu từ kết quả sau khi băm. Việc thay đổi một kí tự trong dữ liệu vào sẽ cho ra một chuỗi kí tự khác rất nhiều so với chuỗi kí tự ban đầu. Tuy nhiên, do việc dữ liệu ra bị giới hạn về độ dài nên chắc chắn sẽ không tránh được việc 2 dữ liệu vào khác nhau cho ra cùng một dữ liệu ra.

SHA-256 (Secure Hash Algorithm 256) là một hàm băm mã hóa được sử dụng rộng rãi trong mã hóa và bảo mật thông tin. Nó thuộc họ thuật toán băm SHA-2, được thiết kế bởi Cơ quan An ninh Quốc gia Hoa Kỳ (NSA) và được phát triển bởi Viện Tiêu chuẩn và Công nghệ Quốc gia (NIST). Đúng như tên gọi, SHA-256 nhận giá trị đầu vào và tạo ra kết quả băm dài 256 bit. SHA-256 được ứng dụng trong rất nhiều lĩnh vực khác nhau như

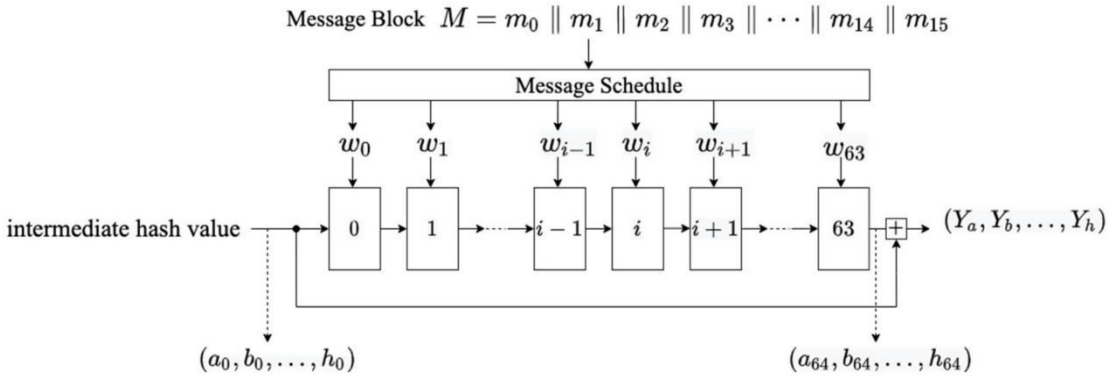
- Bảo mật dữ liệu: SHA-256 tạo kết quả băm duy nhất cho dữ liệu, giúp xác định tính toàn vẹn của dữ liệu. Bất kỳ thay đổi nào trong dữ liệu sẽ dẫn đến thay đổi kết quả ban đầu, giúp phát hiện sự thay đổi trái phép.
- Xác thực mật khẩu: Lưu trữ mật khẩu dưới dạng mã hash sẽ giúp bảo vệ mật khẩu gốc. Khi đăng nhập, mật khẩu nhập vào sẽ được băm và so sánh với kết quả băm đã được lưu trữ. Việc này sẽ giúp nâng cao bảo mật, ngăn chặn tiết lộ mật khẩu trong trường hợp hệ thống bị tấn công.
- Chứng thực dữ liệu: Tạo mã hash cho dữ liệu để xác nhận tính toàn vẹn của dữ liệu khi truyền tải. Bên nhận có thể kiểm tra tính chính xác của dữ liệu bằng cách so sánh mã hash.
- Blockchain: SHA-256 tạo mã hash cho các khối dữ liệu trong blockchain. Các khối dữ liệu này được liên kết với nhau tạo thành chuỗi, đảm bảo tính toàn vẹn và chống giả mạo.

2.2.2 Nguyên lý hoạt động của SHA-256

SHA-256 hoạt động dựa trên 2 bước là xử lý dữ liệu đầu vào và băm dữ liệu. Một số kí hiệu được sử dụng ở phần này bao gồm

- $\oplus$ biểu diễn cho toán tử XOR.
- $\wedge$ biểu diễn cho toán tử AND.
- $\neg$ biểu diễn cho toán tử NOT.
- Phép cộng ở đây là cộng nhị phân modulo 2^{32}.

2.2.2.1 Xử lý dữ liệu đầu vào



Hình 2-2. Quá trình xử lý dữ liệu đầu vào của SHA-256

Dữ liệu đầu vào sẽ được chuyển thành chuỗi nhị phân. Sau đó, ta thêm vào chuỗi nhị phân trên một bit ‘1’ và các bit ‘0’ sao cho khi thêm 64 bit là độ

dài của chuỗi nhị phân ban đầu ở dạng nhị phân thì ta được kết quả là một chuỗi nhị phân có độ dài là bội số của 512. Tiếp đến, ta sẽ chia chuỗi nhị phân trên thành các khối m_i có độ dài 32 bit và khởi tạo một mảng w gồm 64 phần tử được xác định như sau: với $0 \leq i \leq 15$ thì $w_i = m_i$ và với $16 \leq i \leq 63$ thì

$$w_i = \sigma_0(w_{i-2}) + w_{i-7} + \sigma_1(w_{i-15}) + w_{i-16}$$

trong đó

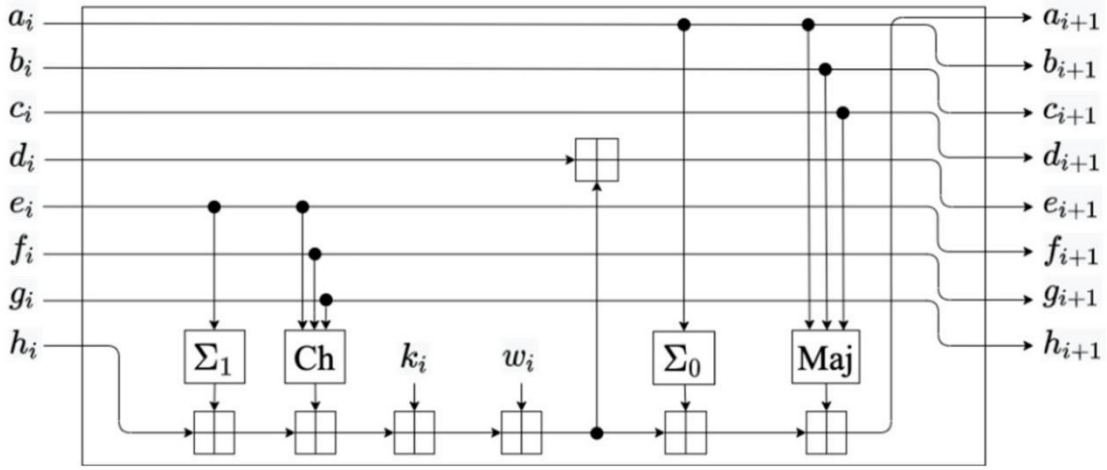
$$\sigma_0(x) = \mathbf{ROTR}(7, x) \oplus \mathbf{ROTR}(18, x) \oplus \mathbf{SHR}(3, x)$$

và

$$\sigma_1(x) = \mathbf{ROTR}(17, x) \oplus \mathbf{ROTR}(9, x) \oplus \mathbf{SHR}(10, x)$$

với $\mathbf{ROTR}(n, x)$ là phép quay phải $n-$bit chuỗi nhị phân x, còn $\mathbf{SHR}(n, x)$ là phép dịch phải $n-$bit chuỗi nhị phân x.

2.2.2.2 Mã hóa dữ liệu



Hình 2-3. Quá trình mã hóa dữ liệu của SHA-256

Trong quá trình mã hóa dữ liệu của SHA-256 có sử dụng đến các đại lượng sau:

- Mảng giá trị băm h_const gồm 8 phần tử là 32 bit đầu tiên trong phần thập phân của căn bậc 2 của 8 số nguyên tố đầu tiên.

Biến	***Giá trị hệ nhị phân***	***Giá trị hệ 16***
h_const_0	01101010000100111100110011001100	6a09e667
h_const_1	10111011011001111010111010000101	bb67ae85
h_const_2	00111100011011011111001101110010	3c6ef372
h_const_3	10100101010011111111010100111010	a54ff53a

h_const_4	01010001000011100101001001111111	510e527f
h_const_5	10011011000001010110100010001100	9b05688c
h_const_6	00011111100000111101100110101011	1f83d9ab
h_const_7	01011011111000001100110100011001	5be0cd19

Bảng 2-2. Bảng giá trị của h_const

- Mảng hằng số k_values gồm 64 phần tử là 32 bit đầu tiên trong phần thập phân của căn bậc 3 của 64 số nguyên tố đầu tiên.

Biến	***Giá trị hệ nhị phân***	***Giá trị hệ 16***
k_values_0	01000010100010100010111110011000	428a2f98
k_values_1	01110001001101110100010010010001	71374491
k_values_2	10110101110000001111101111001111	b5c0fbcf
k_values_3	11101001101101011101101110100101	e9b5dba5
k_values_4	00111001010101101100001001011011	3956c25b
k_values_5	01011001111100010001000111110001	59f111f1
k_values_6	10010010001111111000001010100100	923f82a4
k_values_7	10101011000111000101111011010101	ab1c5ed5
k_values_8	11011000000001111010101010011000	d807aa98
k_values_9	00010010100000110101101100000001	12835b01
k_values_{10}	00100100001100011000010110111110	243185be
k_values_{11}	01010101000011000111110111000011	550c7dc3
k_values_{12}	01110010101111100101110101110100	72be5d74
k_values_{13}	10000000110111101011000111111110	80deb1fe
k_values_{14}	10011011110111000000011010100111	9bdc06a7
k_values_{15}	11000001100110111111000101110100	c19bf174
k_values_{16}	11100100100110110110100111000001	e49b69c1
k_values_{17}	11101111101111100100011110000110	efbe4786
k_values_{18}	00001111110000011001110111000110	0fc19dc6
k_values_{19}	00100100000011001010000111001100	240ca1cc
k_values_{20}	00101101111010010010110001101111	2de92c6f
k_values_{21}	01001010011101001000010010101010	4a7484aa

k_values_{22}	01011100101100001010100111011100	5cb0a9dc
k_values_{23}	01110110111110011000100011011010	76f988da
k_values_{24}	10011000001110100101000101010010	983a5152
k_values_{25}	10101000001100011100011001101101	a831c66d
k_values_{26}	10110000000000110010011111001000	b00327c8
k_values_{27}	10111111010110010111111111000111	bf597fc7
k_values_{28}	11000110111000000000101111110011	c6e00bf3
k_values_{29}	11010101101001111001000101000111	d5a79147
k_values_{30}	00000110110010010110001101010001	06ca6351
k_values_{31}	00010100001010010010100101100111	14292967
k_values_{32}	00100111101101110000101010000101	27b70a85
k_values_{33}	00101110000110110010000100111000	2e1b2138
k_values_{34}	01001101001011000110110111111100	4d2c6dfc
k_values_{35}	01010011001110000000110100010011	53380d13
k_values_{36}	01100101000010100111001101010100	650a7354
k_values_{37}	01110110011010100000101010111011	766a0abb
k_values_{38}	10000001110000101100100100101110	81c2c92e
k_values_{39}	10010010011100100010110010000101	92722c85
k_values_{40}	10100010101111111110100010100001	a2bfe8a1
k_values_{41}	10101000000110100110011001001011	a81a664b
k_values_{42}	11000010010010111000101101110000	c24b8b70
k_values_{43}	11000111011011000101000110100011	c76c51a3
k_values_{44}	11010001100100101101100000011001	d192d819
k_values_{45}	11010110100110010000110000100100	d6990624
k_values_{46}	11110100000011100011010110000101	f40e3585
k_values_{47}	00010000011010101010000001110000	106aa070
k_values_{48}	00011001101001001100000100010110	19a4c116
k_values_{49}	00011110001101110110110000001000	1e376c08
k_values_{50}	00100111010010000111011101001100	2748774c
k_values_{51}	00110100101100001011110010110101	34b0bcb5

k_values_{52}	00111001000111000000110010110011	391c0cb3
k_values_{53}	01001110110110001010101001001010	4ed8aa4a
k_values_{54}	01011011100111001100101001001111	5b9cca4f
k_values_{55}	01101000001011100110111111110011	682e6ff3
k_values_{56}	01110100100011111000001011101110	748f82ee
k_values_{57}	01111000101001010110001101101111	78a5636f
k_values_{58}	10000100110010000111100000010100	84c87814
k_values_{59}	10001100110001110000001000001000	8cc70208
k_values_{60}	10010000101110101111111111111010	90befffa
k_values_{61}	10100100010100000110110011101011	a4506ceb
k_values_{62}	10111110111110011010001111110111	bef9a3f7
k_values_{63}	11000110011100010111100011110010	c67178f2

Bảng 2-3. Bảng giá trị của k_values

- Các biến a, b, c, d, e, f, g, h được khởi tạo với giá trị ban đầu lần lượt từ h_const_0 đến h_const_7.

Ta sẽ thực hiện 1 vòng lặp với i chạy từ 0 đến 63, trong đó tại mỗi bước i ta sẽ tính toán như sau

a. Khai báo các biến

Ta khai báo các biến như sau

$$\Sigma_1 = \mathbf{ROTR}(6, e) \oplus \mathbf{ROTR}(11, e) \oplus \mathbf{ROTR}(25, e)$$

$$Ch = (e \wedge f) \oplus ((\neg e) \wedge g)$$

$$temp_1 = h + \Sigma_1 + Ch + k\_values_i + w_i$$

$$\Sigma_0 = \mathbf{ROTR}(2, a) \oplus \mathbf{ROTR}(13, a) \oplus \mathbf{ROTR}(22, a)$$

$$Maj = (a \wedge b) \oplus (b \wedge c) \oplus (a \wedge c)$$

$$temp_2 = \Sigma_0 + Maj$$

b. Cập nhật giá trị

Ta cập nhật giá trị các biến như sau:

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + temp_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = temp_1 + temp_2$$

Sau khi kết thúc vòng lặp, ta cập nhật lại giá trị của h_values như sau

$$h\_const_0 = h\_const_0 + a$$

$$h\_const_1 = h\_const_1 + b$$

$$h\_const_2 = h\_const_2 + c$$

$$h\_const_3 = h\_const_3 + d$$

$$h\_const_4 = h\_const_4 + e$$

$$h\_const_5 = h\_const_5 + f$$

$$h\_const_6 = h\_const_6 + g$$

$$h\_const_7 = h\_const_7 + h$$

2.2.2.3 Kết quả mã hóa

Sau khi kết thúc quá trình mã hóa, kết quả mã hóa chính là giá trị của các biến h_const_i trong hệ 16 rồi nối lại với nhau.

2.3 NGÔN NGỮ LẬP TRÌNH PYTHON

2.3.1 Giới thiệu về ngôn ngữ Python

Python là một ngôn ngữ lập trình mạnh mẽ và linh hoạt, được sử dụng rộng rãi trong rất nhiều lĩnh vực như phát triển web, phân tích dữ liệu, trí tuệ nhân tạo, tự động hóa,… Đây là một ngôn ngữ có cú pháp đơn giản, dễ học, dễ sử dụng, được hỗ trợ đa nền tảng, có nguồn thư viện phong phú cũng như cộng đồng sử dụng lớn.

2.3.2 Thư viện Tkinter trong Python

Tkinter là một thư viện tiêu chuẩn đi kèm trong Python, được sử dụng để tạo giao diện đồ họa người dùng (GUI). Nó cung cấp một loạt các widget như nút, hộp văn bản, hộp kiểm, thanh cuộn, khung,… để xây dựng các ứng dụng giao diện người dùng trực quan.

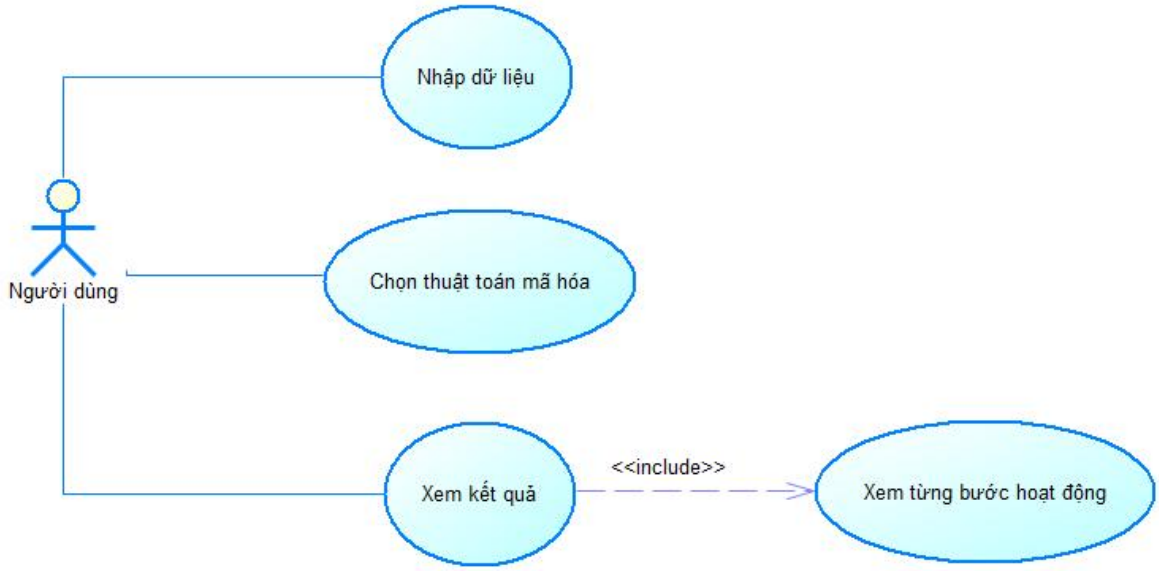
Chương 3. PHÂN TÍCH THIẾT KẾ CHƯƠNG TRÌNH

3.1 SƠ ĐỒ USE-CASE

3.1.1 Các tác nhân

Ở đây chỉ có một tác nhân duy nhất là người dùng. Người sử dụng có thể tự nhập nội dung cần mã hóa hoặc đưa 1 file có nội dung cần mã hóa vào hệ thống. Tiếp đến, người dùng có thể chọn thuật toán mã hóa dữ liệu tương ứng (RSA hoặc SHA-256) rồi xem kết quả. Đối với mỗi thuật toán mã hóa, người dùng có thể xem từng bước hoạt động của nó để có cái nhìn trực quan hơn về thuật toán mã hóa đó.

3.1.2 Các Use-Case



Hình 3-1. Sơ đồ Use-Case

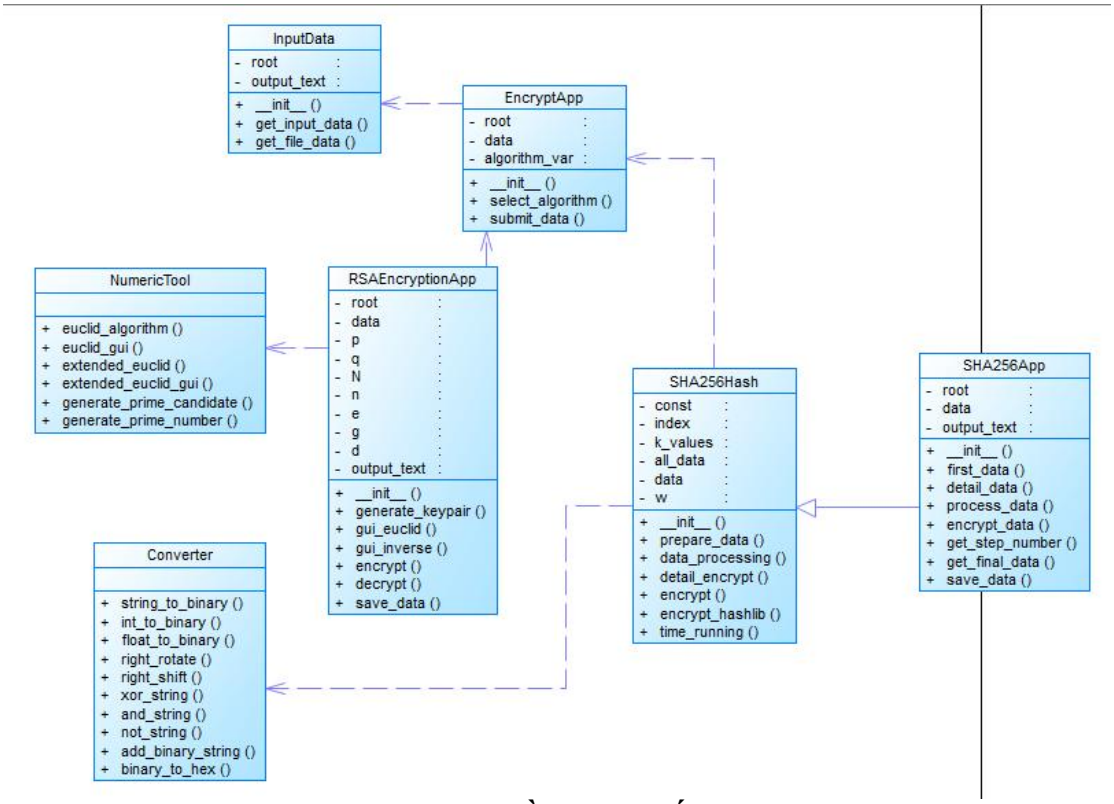
- Nhập dữ liệu: Người dùng có thể nhập dữ liệu trực tiếp hoặc sử dụng file.
- Chọn cách mã hóa: Người dùng có thể chọn một trong hai cách mã hóa là RSA hoặc hàm băm SHA-256.
- Hiển thị kết quả: Người dùng có thể xem kết quả sau khi mã hóa cũng như từng bước hoạt động của thuật toán mã hóa mà người dùng chọn.

3.2 XÂY DỰNG CÁC LỚP ĐỐI TƯỢNG

3.2.1 Xác định các lớp đối tượng

- InputData: Dùng để lấy dữ liệu trực tiếp hoặc từ file.
- EncryptApp: Quản lý các chức năng chính, bao gồm việc lấy dữ liệu từ việc nhập trực tiếp hoặc lấy từ file văn bản, chọn thuật toán mã hóa, hiển thị kết quả cũng như chi tiết hoạt động của từng bước.
- Converter: Chuyển dữ liệu thành dạng chuỗi nhị phân và chuỗi hệ 16 cũng như xử lý các toán tử nhị phân.
- NumericTool: Thực hiện xử lý thuật toán Euclid và Euclid mở rộng.
- SHA256Hash: Xử lý việc thực hiện băm dữ liệu bằng thuật toán băm SHA-256.
- SHA256App (kế thừa lớp SHA256Hash): Trực quan từng bước hoạt động của SHA-256 trên giao diện.
- RSAEncryptionApp: Xử lý việc mã hóa dữ liệu bằng thuật toán mã hóa RSA cũng như trực quan từng bước hoạt động trên giao diện.

3.2.2 Thiết kế lớp chi tiết



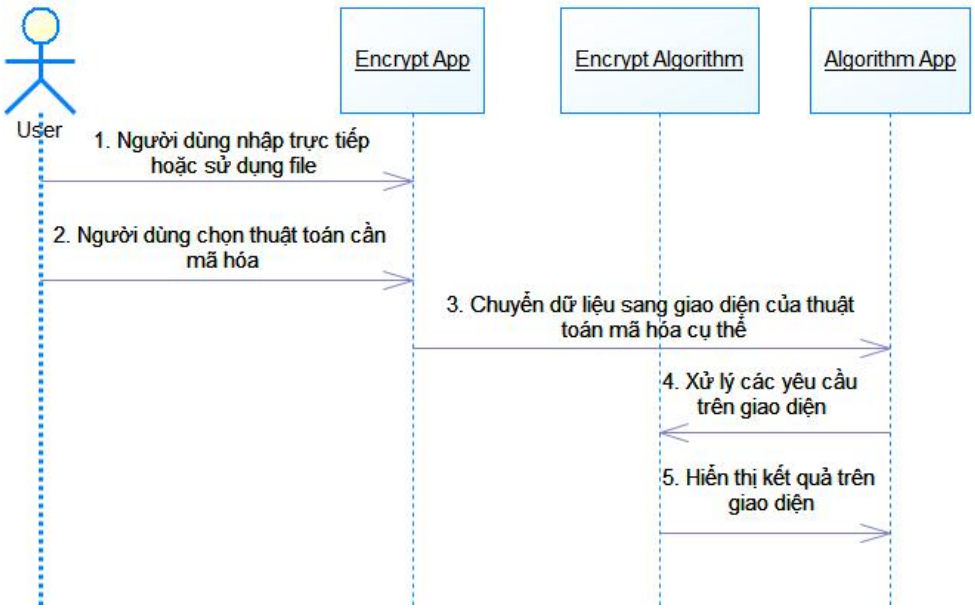
Hình 3-2. Sơ đồ các lớp đối tượng

3.3 SƠ ĐỒ TRÌNH TỰ

Với mỗi ca sử dụng, cần phải xây dựng một biểu đồ trình tự mô tả việc trao đổi thông điệp giữa các đối tượng dọc theo trục thời gian.

Ví dụ ca sử dụng : khi có người dùng có yêu cầu sử dụng ứng dụng thì trình tự công việc phải thực hiện như sau:

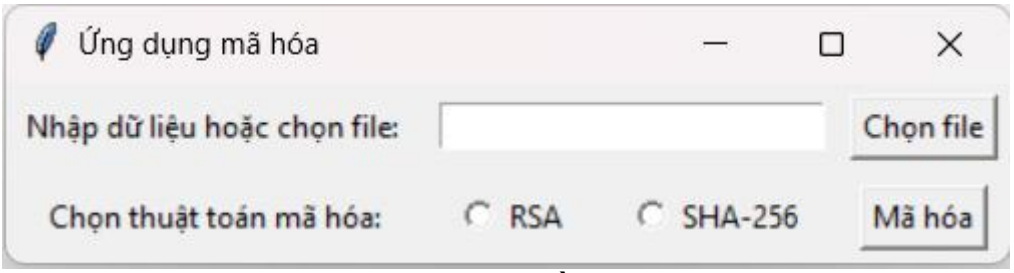
1. Người dùng nhập dữ liệu trực tiếp hoặc sử dụng file có dữ liệu.
2. Người dùng chọn thuật toán cần mã hóa.
3. Hệ thống sẽ thực hiện hành động và trả về giao diện chi tiết từng bước hoạt động của thuật toán mã hóa mà người dùng đã chọn.
4. Người dùng thao tác các chức năng trên giao diện mới.



Hình 3-3. Sơ đồ trình tự xử lý yêu cầu

3.4 GIAO DIỆN NGƯỜI DÙNG

3.4.1 Giao diện ban đầu



Hình 3-4. Giao diện ban đầu của người dùng

Trên giao diện này sẽ có các chức năng:

- Vùng nhập dữ liệu cần mã hóa hoặc lấy dữ liệu từ file.
- Vùng chọn thuật toán mã hóa.
- Nút "Mã hóa" để đi vào chi tiết quá trình mã hóa.

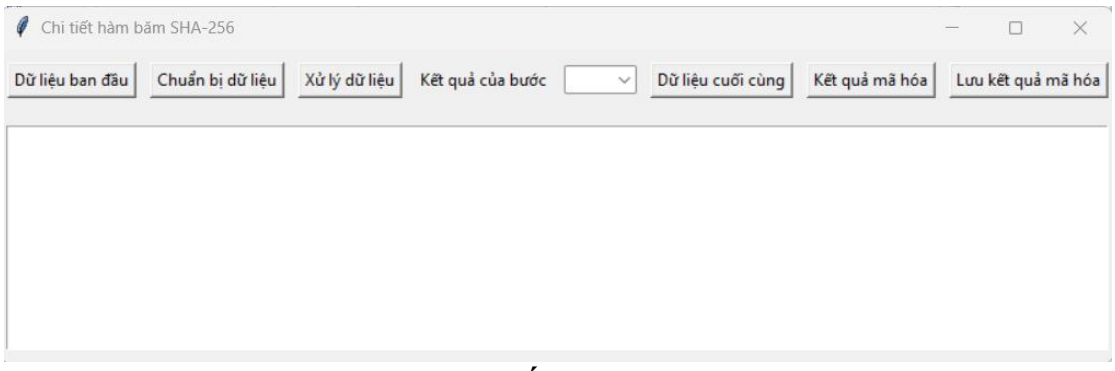
3.4.2 Giao diện chi tiết thuật toán mã hóa RSA

Hình 3-5. Giao diện chi tiết các bước mã hóa của RSA

Trên giao diện này sẽ có các chức năng:

- Ô để nhập độ dài của khóa. Nếu không nhập thì mặc định độ dài là 1024.
- Nút "Sinh cặp số nguyên tố" để tạo ra 2 số nguyên tố lớn p và q.
- Nút "Kiểm tra hai số nguyên tố cùng nhau" để trực quan quá trình tìm ước chung lớn nhất của hai số nguyên bằng thuật toán Euclid.
- Nút "Tìm phần tử nghịch đảo" để trực quan hóa quá trình tìm phần tử nghịch đảo của b modulo a bằng thuật toán Euclid mở rộng.
- Nút "Hiển thị cặp khóa" để hiển thị cặp khóa công khai và khóa bí mật.
- Nút "Lưu khóa" để lưu khóa vào file.
- Nút "Mã hóa" dùng để mã hóa dữ liệu.
- Nút "Lưu kết quả mã hóa" dùng để lưu kết quả mã hóa vào file.
- Nút "Giải mã" để giải mã kết quả mã hóa với dữ liệu và khóa lấy từ file (Ở đây RSA dùng mã hóa chứng thực bằng việc mã hóa bằng khóa bí mật và giải mã bằng khóa công khai).

3.4.3 Giao diện chi tiết thuật toán băm SHA-256



Hình 3-6. Giao diện chi tiết các bước băm của SHA-256

Trên giao diện này có các chức năng:

- Nút “Dữ liệu ban đầu” dùng để hiển thị dữ liệu cần mã hóa đã được chuyển qua dạng nhị phân.
- Nút “Chuẩn bị dữ liệu” để hiển thị dữ liệu ban đầu được chuyển sang để chuẩn bị cho quá trình xử lý.
- Nút “Xử lý dữ liệu” để hiển thị dữ liệu sau khi xử lý và chuẩn bị vào quá trình mã hóa.
- Vùng chọn vòng băm để hiển thị kết quả cuối cùng.
- Nút “Dữ liệu cuối cùng” để hiển thị dữ liệu đã được mã hóa.
- Nút “Mã hóa” để hiển thị kết quả mã hóa cũng như so sánh với việc sử dụng thư viện và thời gian.
- Nút “Lưu kết quả mã hóa” để lưu kết quả mã hóa vào file.

Chương 4. CÀI ĐẶT CHƯƠNG TRÌNH

4.1 CẤU TRÚC SOURCE CODE

```
1    import tkinter as tk
2    from tkinter import ttk, filedialog, messagebox
3    import random
4    from sympy import mod_inverse, isprime
5    import hashlib
6    import time
7    class InputData:...
47
48   class EncryptApp:...
88
89   class Converter:...
168
169  class SHA256Hash:...
281
282  # Class kế thừa từ class SHA256Hash dùng để hiển thị các bước của SHA-256
283  class SHA256App(SHA256Hash):...
411
412  # Class dùng để xử lý cũng như hiển thị các bước của thuật toán Euclid và Euclid mở rộng
413  class NumericTool:...
483
484  class RSAEncryptionApp:...
647
648  if __name__ == "__main__":
649      root = tk.Tk()
650      app = EncryptApp(root)
651      root.mainloop()
```

Hình 4-1. Cấu trúc chương trình

4.2 CODE MẪU MỘT SỐ CHỨC NĂNG CHÍNH

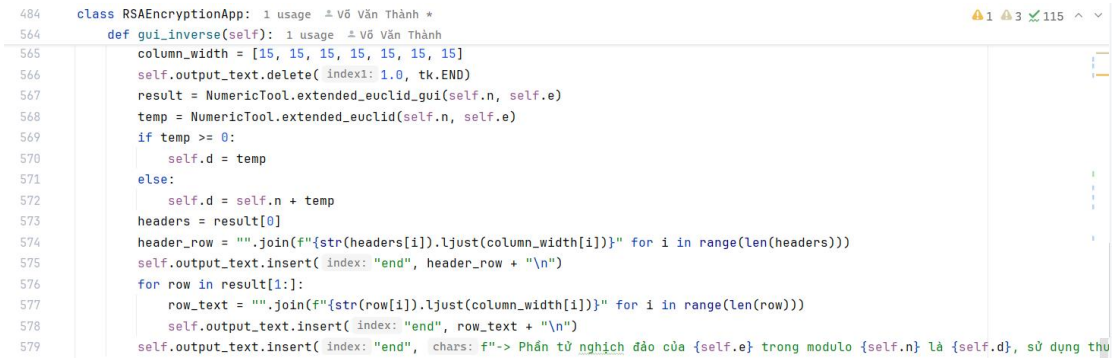
4.2.1 Thuật toán mã hóa RSA

```
484  class RSAEncryptionApp:  1 usage  Võ Văn Thành *
534      def generate_keypair(self):  1 usage  Võ Văn Thành
535          content = self.length.get()
536          length = 1024
537          if content.strip():
538              length = int(content)
539          self.p = NumericTool.generate_prime_number(length)
540          self.q = NumericTool.generate_prime_number(length)
541          self.N = self.p * self.q
542          self.n = (self.p - 1) * (self.q - 1)
543          result = f"Cặp số nguyên tố: ({self.p}, {self.q})\n -> Chuyển sang hệ 16: ({hex(self.p)[2:]}, {hex(self.q)[2:]})\n"
544          self.output_text.delete( index1: 1.0, tk.END)
545          self.output_text.insert(tk.END, result)
```

Hình 4-2. Chức năng sinh cặp số nguyên tố lớn

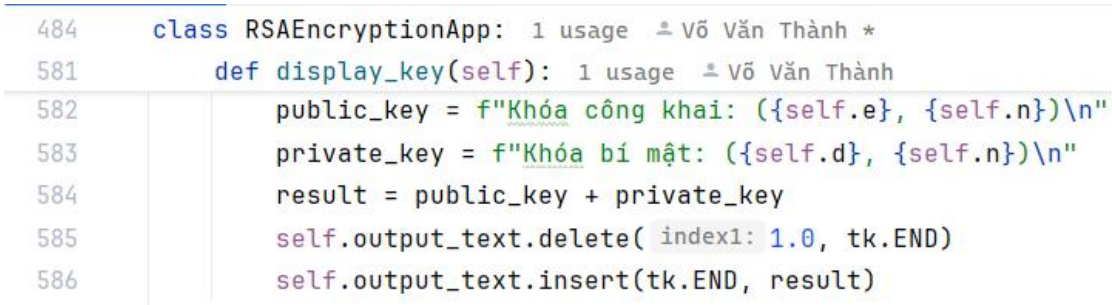
```
484  class RSAEncryptionApp:  1 usage  Võ Văn Thành *
547      def gui_euclid(self):  1 usage  Võ Văn Thành
548          column_width = [20, 20, 20, 20]
549          self.output_text.delete( index1: 1.0, tk.END)
550          self.e = random.randrange( start: 1, self.n)
551          self.g = NumericTool.euclid_algorithm(self.e, self.n)
552          while self.g != 1:
553              self.e = random.randrange( start: 1, self.n)
554              self.g = NumericTool.euclid_algorithm(self.e, self.n)
555          result = NumericTool.euclid_gui(self.e, self.n)
556          headers = result[0]
557          header_row = "".join(f"{str(headers[i]).ljust(column_width[i])}" for i in range(len(headers)))
558          self.output_text.insert( index: "end", header_row + "\n")
559          for row in result[1:]:
560              row_text = "".join(f"{str(row[i]).ljust(column_width[i])}" for i in range(len(row)))
561              self.output_text.insert( index: "end", row_text + "\n")
562          self.output_text.insert( index: "end",  chars: f"-> Ước chung lớn nhất là {self.g} nên {self.e} và {self.n} là hai số nguyên tố cù
```

Hình 4-3. Chức năng trực quan hóa quá trình tìm ước chung lớn nhất của 2 số nguyên tố bằng thuật toán Euclid



```
484  class RSAEncryptionApp:  1 usage  Võ Văn Thành *
564      def gui_inverse(self):  1 usage  Võ Văn Thành
565          column_width = [15, 15, 15, 15, 15, 15, 15]
566          self.output_text.delete( index1: 1.0, tk.END)
567          result = NumericTool.extended_euclid_gui(self.n, self.e)
568          temp = NumericTool.extended_euclid(self.n, self.e)
569          if temp >= 0:
570              self.d = temp
571          else:
572              self.d = self.n + temp
573          headers = result[0]
574          header_row = "".join(f"{str(headers[i]).ljust(column_width[i])}" for i in range(len(headers)))
575          self.output_text.insert( index: "end", header_row + "\n")
576          for row in result[1:]:
577              row_text = "".join(f"{str(row[i]).ljust(column_width[i])}" for i in range(len(row)))
578              self.output_text.insert( index: "end", row_text + "\n")
579          self.output_text.insert( index: "end",  chars: f"-> Phần tử nghịch đảo của {self.e} trong modulo {self.n} là {self.d}, sử dụng thu
```

Hình 4-4. Chức năng trực quan hóa quá trình tìm phần tử nghịch đảo modulo bằng thuật toán Euclid mở rộng và so sánh kết quả với thư viện



```
484  class RSAEncryptionApp:  1 usage  Võ Văn Thành *
581      def display_key(self):  1 usage  Võ Văn Thành
582          public_key = f"Khóa công khai: ({self.e}, {self.n})\n"
583          private_key = f"Khóa bí mật: ({self.d}, {self.n})\n"
584          result = public_key + private_key
585          self.output_text.delete( index1: 1.0, tk.END)
586          self.output_text.insert(tk.END, result)
```

Hình 4-5. Chức năng hiển thị cặp khóa công khai và khóa bí mật

```
484     class RSAEncryptionApp:  1 usage  Võ Văn Thành *
615         def save_data(self):  1 usage  Võ Văn Thành *
630             print("Hủy lưu file.")
631
632         def save_key(self):  1 usage  new *
633             file_path = filedialog.asksaveasfilename(
634                 title = "Lưu file",
635                 defaultextension = ".txt",
636                 filetypes = [("Text Files", "*.txt"), ("All Files", "*.*")]
637             )
638             if file_path:  # Nếu người dùng chọn file
639                 try:
640                     with open(file_path, 'w', encoding='utf-8') as file:
641                         file.write(f"{self.e}\n{self.N}")
642                     print(f"Khóa đã được lưu vào {file_path}")
643                 except Exception as e:
644                     print(f"Lỗi khi lưu file: {e}")
645             else:
646                 print("Hủy lưu file.")
```

Hình 4-6. Chức năng lưu khóa vào file

```
484   class RSAEncryptionApp:  1 usage  Võ Văn Thành *
588       def encrypt(self):  1 usage  Võ Văn Thành
589           start_time = time.time()
590           self.encrypt_message = [hex(pow(ord(char), self.d, self.N))[2:] for char in self.data]
591           end_time = time.time()
592           time_running = end_time - start_time
593           result = f"Kết quả mã hóa: {''.join(self.encrypt_message)}\nThời gian mã hóa: {time_running} giây"
594           self.output_text.delete( index1: 1.0, tk.END)
595           self.output_text.insert(tk.END, result)
```

Hình 4-7. Chức năng hiển thị kết quả mã hóa dữ liệu và thời gian chạy

```
484     class RSAEncryptionApp:  1 usage  Võ Văn Thành *
615         def save_data(self):  1 usage  Võ Văn Thành *
616             file_path = filedialog.asksaveasfilename(
617                 title = "Lưu file",
618                 defaultextension = ".txt",
619                 filetypes = [("Text Files", "*.txt"), ("All Files", "*.*")]
620             )
621             if file_path:  # Nếu người dùng chọn file
622                 try:
623                     with open(file_path, 'w', encoding='utf-8') as file:
624                         for item in self.encrypt_message:
625                             file.write(f"{item}\n")
626                     print(f"Dữ liệu đã được lưu vào {file_path}")
627                 except Exception as e:
628                     print(f"Lỗi khi lưu file: {e}")
629             else:
630                 print("Hủy lưu file.")
```

Hình 4-8. Chức năng lưu kết quả mã hóa vào file

```
484  class RSAEncryptionApp:  1 usage  Võ Văn Thành *
597      def decrypt(self):  1 usage  Võ Văn Thành *
598          start_time = time.time()
599          file_data_path = filedialog.askopenfilename()
600          if file_data_path:
601              with open(file_data_path, 'r', encoding = 'utf-8') as file:
602                  self.encrypt_message = [line.strip() for line in file]
603          file_key_path = filedialog.askopenfilename()
604          if file_key_path:
605              with open(file_key_path, 'r', encoding = 'utf-8') as file:
606                  self.e = int(file.readline().strip())
607                  self.N = int(file.readline().strip())
608          self.decrypt_message = [chr(pow(int(char, 16), self.e, self.N)) for char in self.encrypt_message]
609          end_time = time.time()
610          time_running = end_time - start_time
611          result = f"Kết quả giải mã: {''.join(self.decrypt_message)}\nThời gian giải mã: {time_running} giây"
612          self.output_text.delete( index1: 1.0, tk.END)
613          self.output_text.insert(tk.END, result)
```

Hình 4-9. Chức năng hiển thị kết quả giải mã với dữ liệu và khóa lấy từ file và thời gian chạy

4.2.2 Thuật toán băm SHA-256

```
283  class SHA256App(SHA256Hash):  1 usage  Võ Văn Thành *
322      def first_data(self):  1 usage  Võ Văn Thành *
323          result = Converter.string_to_binary(self.data) + '10000000'
324          length = len(result)
325          result = result.ljust((length + 64 + 511) // 512 * 512 - 64,  __fillchar: '0')
326          result += bin(len(self.data.encode('utf-8')) * 8)[2:].zfill(64)
327          data_binary = [result[32 * i: 32 * (i + 1)] for i in range(16)]
328          temp = ""
329          for i in range(16):
330              temp += f"{data_binary[i]}\n"
331          self.output_text.delete( index1: 1.0, tk.END)
332          self.output_text.insert(tk.END, temp)
```

Hình 4-10. Chức năng hiển thị dữ liệu ban đầu dưới dạng nhị phân

```
283  class SHA256App(SHA256Hash):  1 usage  Võ Văn Thành *
334      def detail_data(self):  1 usage  Võ Văn Thành
335          self.prepare_data()
336          result = ""
337          for i in range(64):
338              result += f"w[{i}]: {self.w[i]}\n"
339          self.output_text.delete( index1: 1.0, tk.END)
340          self.output_text.insert(tk.END, result)
```

Hình 4-11. Chức năng hiển thị dữ liệu chuẩn bị xử lý

```
283     class SHA256App(SHA256Hash):  1 usage  Võ Văn Thành *
342         def process_data(self):  1 usage  Võ Văn Thành
343             self.prepare_data()
344             self.data_processing()
345             result = ""
346             for i in range(64):
347                 result += f"w[{i}]: {self.w[i]}\n"
348             self.output_text.delete( index1: 1.0, tk.END)
349             self.output_text.insert(tk.END, result)
```

Hình 4-12. Chức năng hiển thị dữ liệu đã được xử lý

```
283     class SHA256App(SHA256Hash):  1 usage  Võ Văn Thành *
363         # Function hiển thị kết quả của từng vòng băm
364         def get_step_number(self, event):  1 usage  Võ Văn Thành
365             number = int(self.step_number.get())
366             result = ""
367             sha256_hash = SHA256Hash(self.data)
368             sha256_hash.encrypt()
369             index = sha256_hash.all_data[number - 1]
370             temp = [index[32*i: 32*(i+1)] for i in range(8)]
371             for i in range(8):
372                     result += f"index[{i}]: {temp[i]}\n"
373             self.output_text.delete( index1: 1.0, tk.END)
374             self.output_text.insert(tk.END, result)
```

Hình 4-13. Chức năng hiển thị kết quả của từng vòng băm

```
283  class SHA256App(SHA256Hash):  1 usage  Võ Văn Thành *                                                    1  3  115
376      def final_data(self):  1 usage  Võ Văn Thành *
377          sha256_variable = SHA256Hash(self.data)
378          sha256_variable.prepare_data()
379          sha256_variable.data_processing()
380          sha256_variable.const = [bin(int(x, 16))[2:].zfill(32) for x in sha256_variable.const]
381          sha256_variable.k_values = [bin(x)[2:].zfill(32) for x in sha256_variable.k_values]
382          for i in range(8):
383              sha256_variable.index.append(sha256_variable.const[i])
384          for i in range(64):
385              sha256_variable.detail_encrypt(i)
386          for i in range(8):
387              sha256_variable.const[i] = Converter.add_binary_string( *strings: sha256_variable.const[i], sha256_variable.index[i])
388          result = ""
389          for i in range(8):
390              result += f"const[{i}]: {sha256_variable.const[i]} -> Chuyển sang hệ 16: {Converter.binary_to_hex(sha256_variable.const[i])}
391          self.output_text.delete( index1: 1.0, tk.END)
392          self.output_text.insert(tk.END, result)
```

Hình 4-14. Chức năng hiển thị kết quả mã hóa

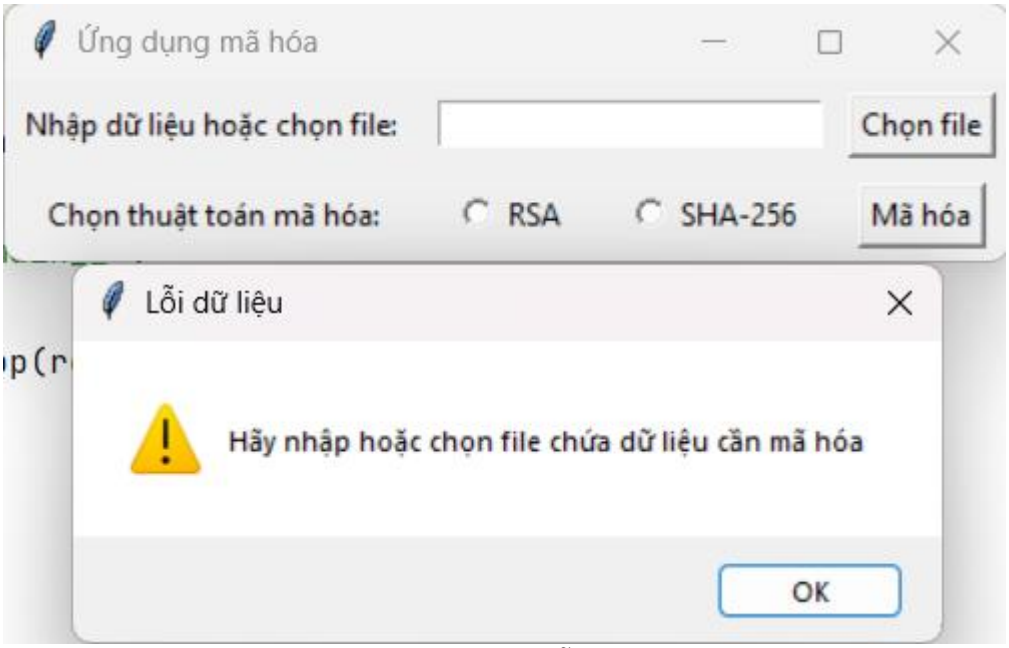
```
283  class SHA256App(SHA256Hash):  1 usage  Võ Văn Thành *
351      def encrypt_data(self):  1 usage  Võ Văn Thành
352          sha256_obj = SHA256Hash(self.data)
353          sha256_obj.encrypt()
354          hashed_message_library = SHA256Hash.encrypt_hashlib(self.data)
355          time1, time2 = SHA256Hash.time_running(self.data)
356          str1 = f"Kết quả mã hóa: \n"
357          str2 = f"Không sử dụng thư viện: {sha256_obj.encrypt_message}\nThời gian thực hiện: {time1} giây\n"
358          str3 = f"Sử dụng thư viện: {hashed_message_library}\nThời gian thực hiện: {time2} giây"
359          result = str1 + str2 + str3
360          self.output_text.delete( index1: 1.0, tk.END)
361          self.output_text.insert(tk.END, result)
```

Hình 4-15. Chức năng hiển thị dữ liệu đã được mã hóa và so sánh với thư viện

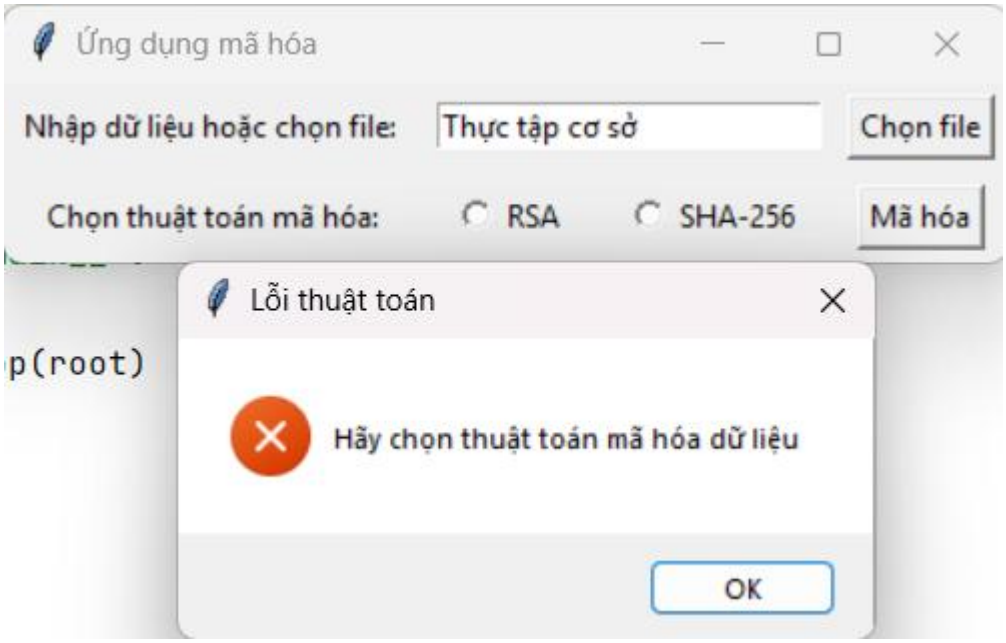
```
283  class SHA256App(SHA256Hash):  1 usage  Võ Văn Thành *
394      def save_data(self):  1 usage  Võ Văn Thành
395          file_path = filedialog.asksaveasfilename(
396              title = "Lưu file",
397              defaultextension = ".txt",
398              filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
399          )
400          sha256_obj = SHA256Hash(self.data)
401          sha256_obj.encrypt()
402          if file_path:  # Nếu người dùng chọn file
403              try:
404                  with open(file_path, 'w', encoding='utf-8') as file:
405                      file.write(sha256_obj.encrypt_message)
406                  print(f"Dữ liệu đã được lưu vào {file_path}")
407              except Exception as e:
408                  print(f"Lỗi khi lưu file: {e}")
409          else:
410              print("Hủy lưu file.")
```

Hình 4-16. Chức năng lưu kết quả mã hóa vào file

Chương 5. THỰC THI CHƯƠNG TRÌNH

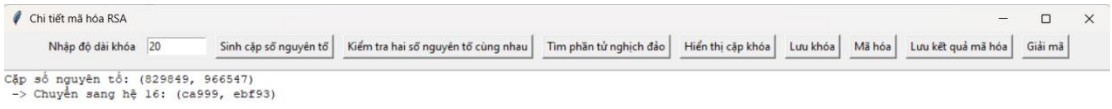


Hình 5-1. Giao diện báo lỗi chưa có dữ liệu

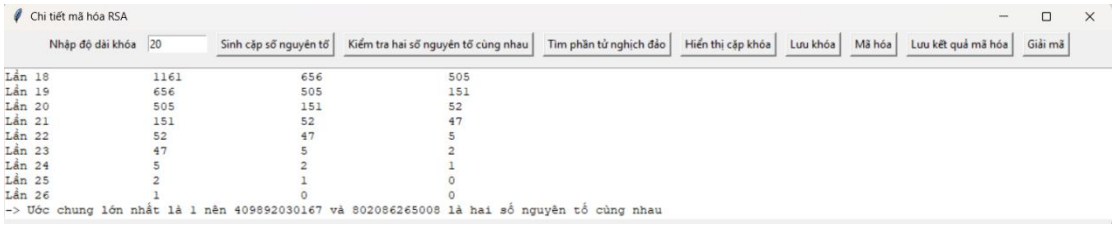


Hình 5-2. Giao diện báo lỗi chưa chọn thuật toán mã hóa

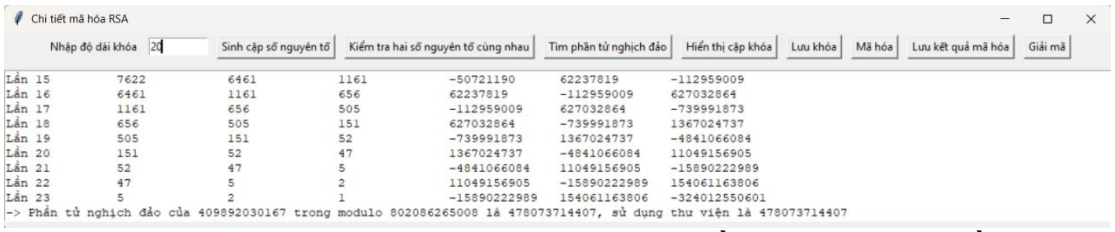
5.1 THUẬT TOÁN MÃ HÓA RSA



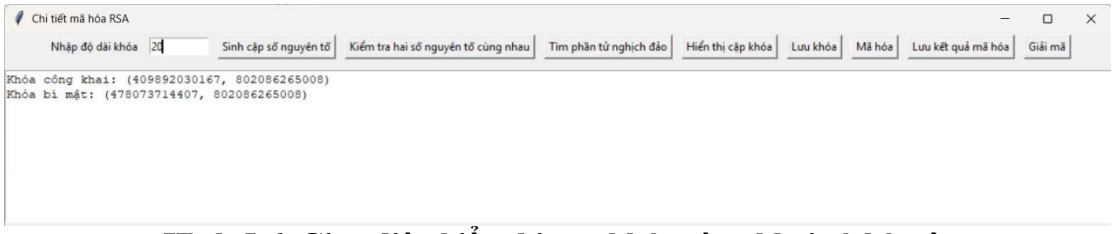
Hình 5-3. Giao diện hiển thị cặp số nguyên tố lớn với độ dài nhập từ bàn phím



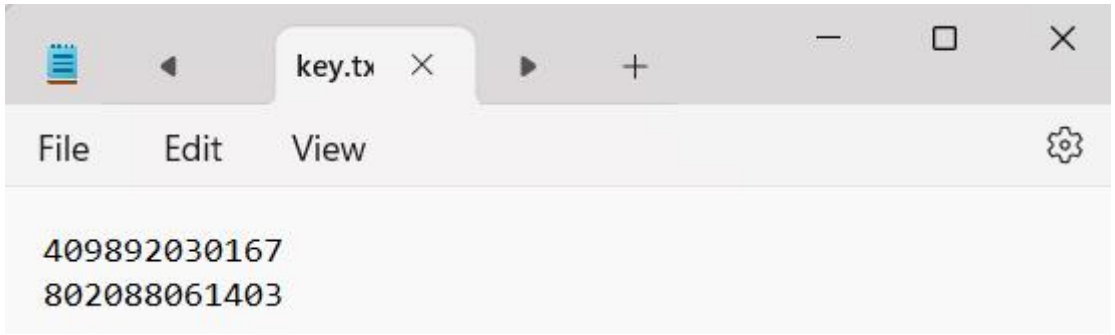
Hình 5-4. Giao diện trực quan hóa quá trình tìm ước chung lớn nhất của hai số nguyên bằng thuật toán Euclid



Hình 5-5. Giao diện trực quan hóa quá trình tìm phần tử nghịch đảo bằng thuật toán Euclid mở rộng

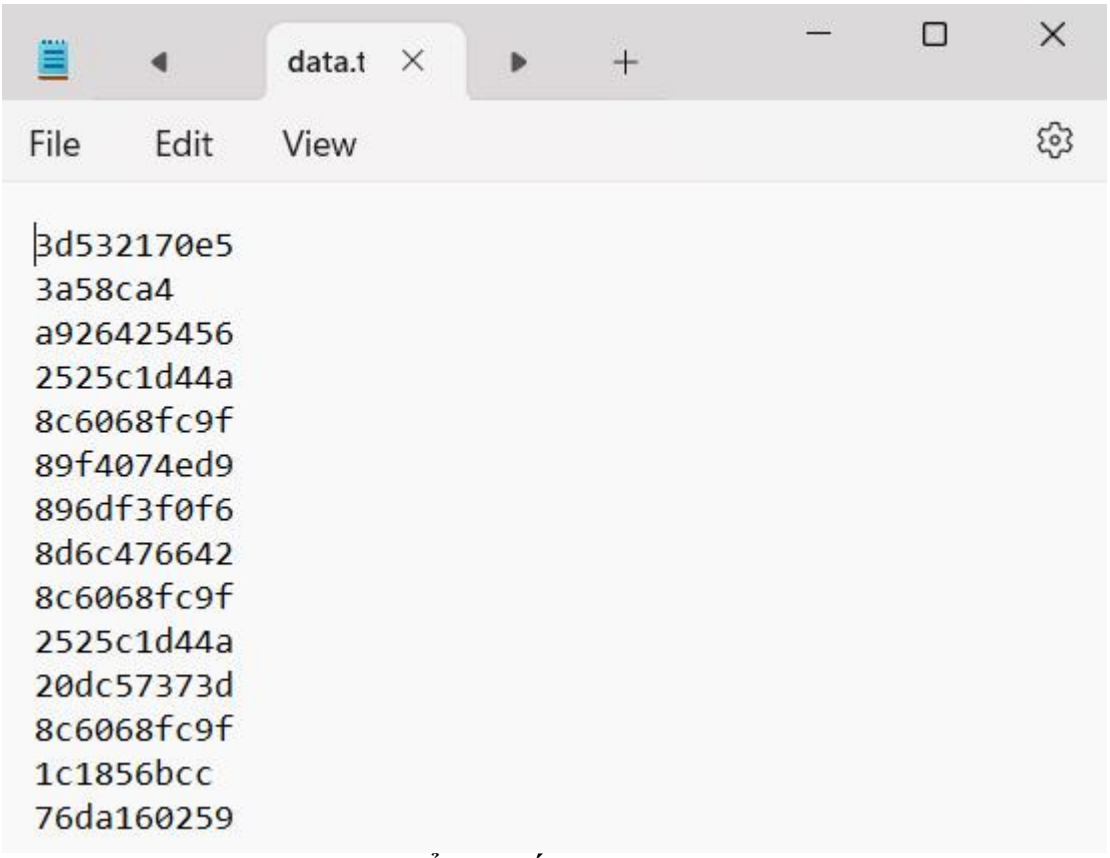


Hình 5-6. Giao diện hiển thị cặp khóa công khai và bí mật

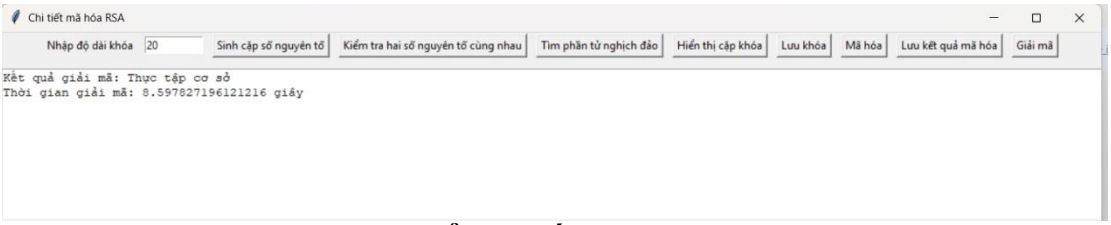


Hình 5-7. Giao diện hiển thị khóa công khai được lưu trong file

Hình 5-8. Giao diện hiển thị kết quả mã hóa và thời gian chạy

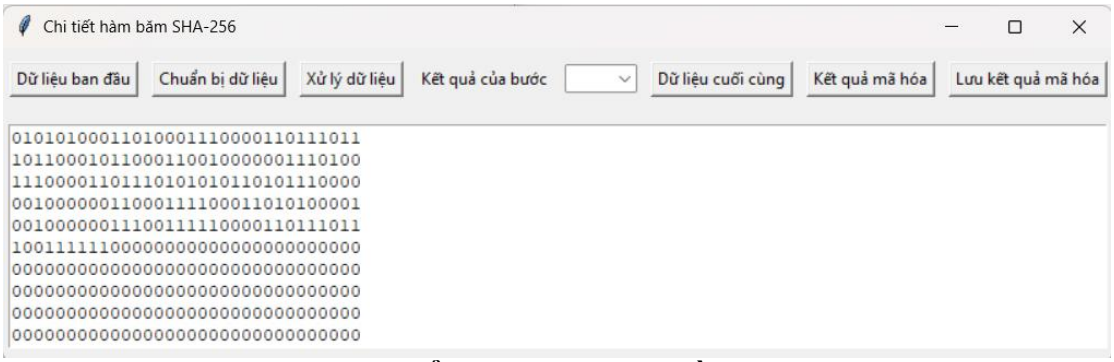


Hình 5-9. Giao diện hiển thị kết quả mã hóa được lưu trong file

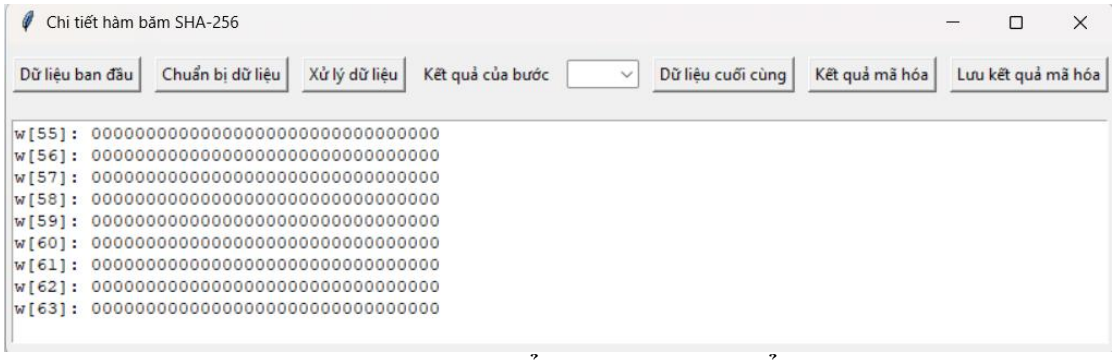


Hình 5-10. Giao diện hiển thị kết quả giải mã và thời gian chạy

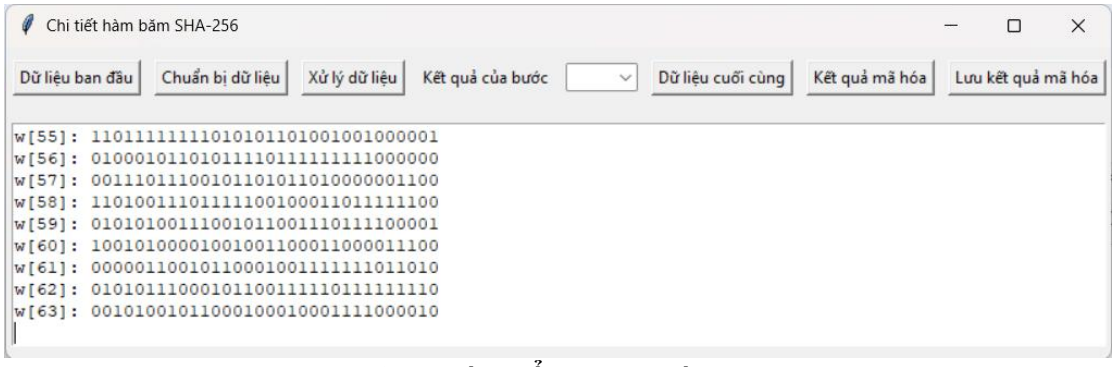
5.2 THUẬT TOÁN BĂM SHA-256



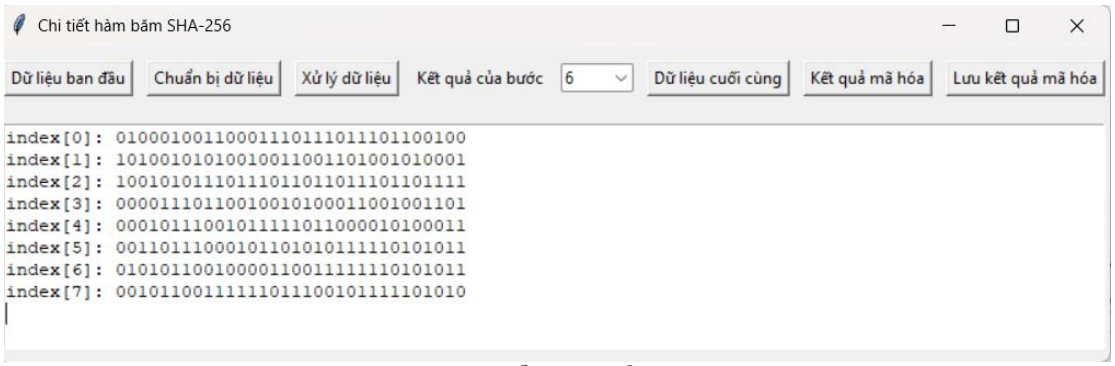
Hình 5-11. Giao diện hiển thị dữ liệu ban đầu dưới dạng nhị phân



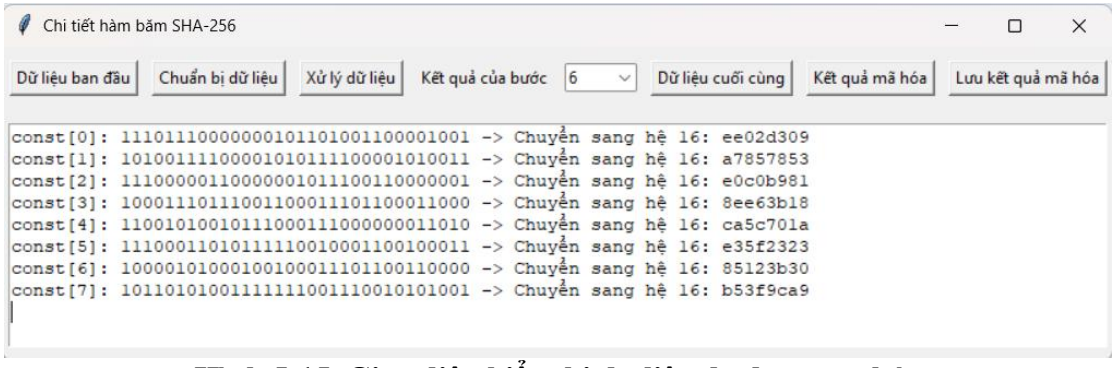
Hình 5-12. Giao diện hiển thị dữ liệu chuẩn bị xử lý



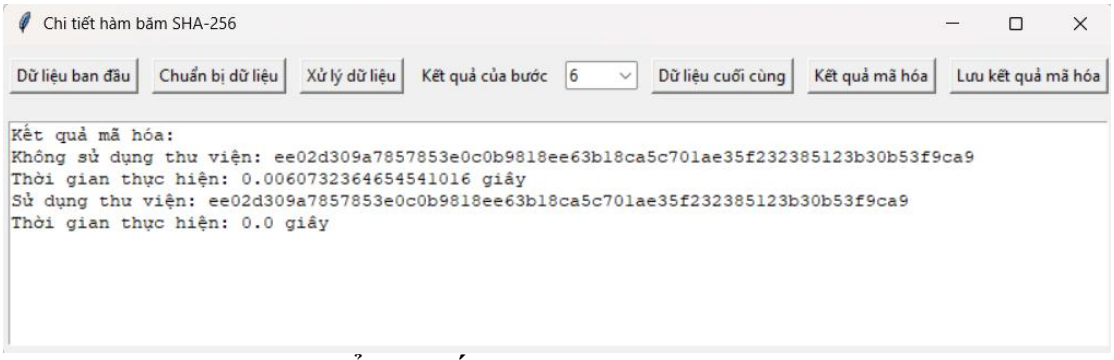
Hình 5-13. Giao diện hiển thị dữ liệu đã được xử lý



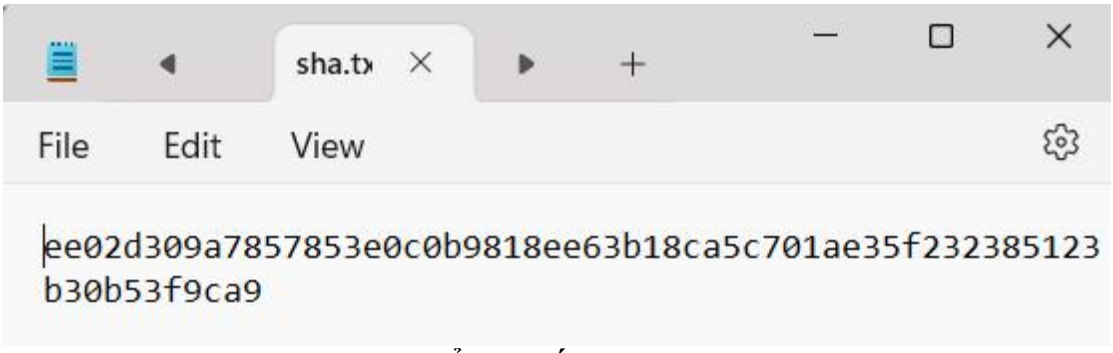
Hình 5-14. Giao diện hiển thị kết quả từng vòng băm



Hình 5-15. Giao diện hiển thị dữ liệu đã được mã hóa



Hình 5-16. Giao diện hiển thị kết quả mã hóa và so sánh với sử dụng thư viện



Hình 5-17. Giao diện hiển thị kết quả mã hóa được lưu vào file

Chương 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 KẾT QUẢ ĐẠT ĐƯỢC

Sau quá trình tìm hiểu, phân tích, thiết kế, cài đặt và triển khai thì em cũng đã hoàn thành được đề tài "Cài đặt thuật toán mã hóa RSA và hàm băm SHA-256 trong Python". Về mặt lý thuyết, em đã nắm được khái niệm cũng như nguyên lý hoạt động của RSA và SHA-256. Về mặt phân tích, em đã nắm được cách phân tích thiết kế theo hướng đối tượng cũng như vẽ các sơ đồ UML. Về mặt triển khai, em đã áp dụng được kỹ thuật lập trình hướng đối tượng cũng như sử dụng thư viện trong Python. Em cũng đã mô phỏng được các thuật toán trên, đã tạo được giao diện người dùng cũng như kiểm tra được tính đúng đắn của các thuật toán.

6.2 NHỮNG HẠN CHẾ

- Giao diện còn đơn giản, chưa có nhiều chức năng.
- Chưa chia đôi giao diện để so sánh kết quả với việc sử dụng thư viện mà dùng chung giao diện.
- Chưa so sánh được hiệu năng cũng như tiêu tốn tài nguyên so với thư viện.
- Phần trực quan thuật toán Euclid và Euclid mở rộng còn đơn giản.
- Đối với RSA thì dữ liệu sẽ có sự sai sót với độ dài khóa nhỏ.
- Đối với SHA-256 thì đôi lúc dữ liệu lấy từ file bị mã hóa sai.

6.3 HƯỚNG PHÁT TRIỂN

Có thể phát triển bằng cách thêm nhiều thuật toán mã hóa khác, so sánh giữa các thuật toán mã hóa (thời gian chạy, độ bảo mật, hiệu năng, bộ nhớ,…); có thể kết hợp với các website cũng như hệ thống để tăng tính bảo mật bằng cách mã hóa mật khẩu,…

TÀI LIỆU THAM KHẢO

1. Phạm Văn Nam, *Bài giảng Lập trình Python*.
2. Trần Minh Văn, *Giáo trình An toàn và bảo mật thông tin*.
3. Lê Thị Bích Hằng - Nguyễn Đình Hưng, *Phân tích và thiết kế hướng đối tượng sử dụng UML*.
4. NIST, U. (1995). Secure hash standard. *Federal Information Processing Standard (FIPS) 180-1*.

PHỤ LỤC: CODE CHI TIẾT CHƯƠNG TRÌNH

Các thư viện sử dụng

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import random
from sympy import mod_inverse, isprime
import hashlib
import time
```

Lớp InputData

```
class InputData:
    def __init__(self, root_index):
        self.root = root_index

        # Tạo label và entry để nhập dữ liệu
        self.label = tk.Label(root_index, text = "Nhập dữ liệu hoặc chọn
file: ")
        self.label.grid(row = 0, column = 0, padx = 5, pady = 5)
        self.entry = tk.Entry(root_index, width = 25)
        self.entry.grid(row = 0, column = 1, columnspan = 2, padx = 5, pady
= 5)

        # Tạo button để chọn file
        self.file_button = tk.Button(root_index, text = "Chọn file",
command = self.get_file_data)
        self.file_button.grid(row = 0, column = 3, padx = 5, pady = 5)

        # Đường dẫn của file cần lấy dữ liệu
        self.file_path = None

    # Function lấy dữ liệu trực tiếp
    def get_input_data(self):
        return self.entry.get()

    # Function lấy dữ liệu từ file
    def get_file_data(self):
        self.file_path = filedialog.askopenfilename()
        if self.file_path:
            try:
```

```
                with open(self.file_path, 'r', encoding='utf-8') as file:
                    content = file.read()
                    self.entry.delete(0, tk.END)
                    self.entry.insert(tk.END, content)
            except UnicodeDecodeError:
                try:
                    with open(self.file_path, 'r', encoding='utf-16') as
file:
                        content = file.read()
                        self.entry.delete(0, tk.END)
                        self.entry.insert(tk.END, content)
                except Exception as e:
                    messagebox.showerror("Lỗi", f"Không thể mở file: {e}")
            except Exception as e:
                messagebox.showerror("Lỗi", f"Không thể mở file: {e}")
```

Lớp EncryptApp

```
class EncryptApp:
    def __init__(self, root_index):
        self.root = root_index
        self.root.title("Ứng dụng mã hóa")

        # Lấy dữ liệu từ class InputData
        self.input_data = InputData(root_index)

        # Tạo mục chọn thuật toán mã hóa
        self.algorithm_label = tk.Label(root_index, text = "Chọn thuật toán
mã hóa:")
        self.algorithm_label.grid(row = 1, column = 0, padx = 5, pady = 5)
        self.algorithm_var = tk.StringVar(value = "Default")
        self.rsa_radio = tk.Radiobutton(root_index, text = "RSA", variable
= self.algorithm_var, value = "RSA")
        self.sha_radio = tk.Radiobutton(root_index, text = "SHA-256",
variable = self.algorithm_var, value = "SHA")
        self.rsa_radio.grid(row = 1, column = 1, padx = 5, pady = 5)
        self.sha_radio.grid(row = 1, column = 2, padx = 5, pady = 5)

        # Tạo button để mã hóa dữ liệu
        self.submit_button = tk.Button(root_index, text = "Mã hóa", command
= self.submit_data)
```

```
        self.submit_button.grid(row = 1, column = 3, padx = 5, pady = 5)

    # Function khi đưa dữ liệu vào mã hóa
    def submit_data(self):
        data = self.input_data.get_input_data()
        algorithm = self.algorithm_var.get()

        if not data:
            messagebox.showwarning("Lỗi dữ liệu", "Hãy nhập hoặc chọn file
chứa dữ liệu cần mã hóa")
            return

        if algorithm == "SHA":
            sha256_root = tk.Tk()
            sha256_app = SHA256App(sha256_root, data)
            sha256_root.mainloop()
        elif algorithm == "RSA":
            rsa_root = tk.Tk()
            rsa_app = RSAEncryptionApp(rsa_root, data)
            rsa_root.mainloop()
        else:
            messagebox.showerror("Lỗi thuật toán", "Hãy chọn thuật toán mã
hóa dữ liệu")
```

Lớp Converter

```
class Converter:
    # Function chuyển từ chuỗi kí tự sang chuỗi nhị phân
    @staticmethod
    def string_to_binary(string):
        return ''.join(format(byte, '08b') for byte in string.encode('utf-
8'))

    # Function chuyển từ số nguyên sang chuỗi nhị phân
    @staticmethod
    def int_to_binary(n):
        return bin(n)[2:].zfill(32)

    # Function chuyển phần thập phân của số thực thành chuỗi nhị phân
    @staticmethod
    def float_to_binary(n, precision = 32):
```

```
        frac_part = n - int(n)
        result = ''
        while precision:
            frac_part *= 2
            bit = int(frac_part)
            if bit == 1:
                frac_part -= bit
                result += '1'
            else:
                result += '0'
            precision -= 1
        return result

    # Function xoay phải chuỗi nhị phân
    @staticmethod
    def right_rotate(string, x):
        return string[-x:] + string[:-x]

    # Function dịch phải chuỗi nhị phân
    @staticmethod
    def right_shift(string, x):
        return '0' * x + string[:-x]

    # Function thực hiện XOR nhiều chuỗi nhị phân
    @staticmethod
    def xor_string(*strings):
        max_len = max(len(s) for s in strings)
        strings = [s.zfill(max_len) for s in strings]
        result = list(strings[0])
        for i in range(1, len(strings)):
            current_str = strings[i]
            for j in range(len(result)):
                result[j] = '1' if result[j] != current_str[j] else '0'
        return ''.join(result)

    # Function thực hiện AND nhiều chuỗi nhị phân
    @staticmethod
    def and_string(*strings):
        max_len = max(len(s) for s in strings)
        strings = [s.zfill(max_len) for s in strings]
```

```
        result = list(strings[0])
        for i in range(1, len(strings)):
            current_str = strings[i]
            for j in range(len(result)):
                result[j] = '1' if result[j] == '1' and current_str[j] ==
'1' else '0'
        return ''.join(result)

    # Function thực hiện NOT một chuỗi nhị phân
    @staticmethod
    def not_string(string):
        return ''.join('1' if ch == '0' else '0' for ch in string)

    # Function cộng nhiều chuỗi nhị phân
    @staticmethod
    def add_binary_string(*strings):
        total = sum(int(s, 2) for s in strings)
        result = total % (2**32)
        return Converter.int_to_binary(result)

    # Function chuển từ nhị phân sang hệ 16
    @staticmethod
    def binary_to_hex(binary_string):
        index = int(binary_string, 2)
        result = hex(index)[2:]
        return result.zfill(8)
```

Lớp SHA256Hash

```
class SHA256Hash:
    def __init__(self, string):
        # Biểu diễn hệ 16 của phần thập phân của căn bậc 2 của 8 số nguyên
tố đầu tiên
        self.const = [
            '6a09e667', 'bb67ae85', '3c6ef372', 'a54ff53a',
            '510e527f', '9b05688c', '1f83d9ab', '5be0cd19'
        ]

        # Mảng dùng để lưu giá trị thay đổi trong lúc băm
        self.index = []
```

```
        # Mảng dùng để lưu tất cả dữ liệu của từng vòng băm
        self.all_data = []

        # Biểu diễn hệ 16 của phần thập phân của căn bậc 3 của 64 số nguyên
tố đầu tiên
        self.k_values = [
            0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
            0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
            0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
            0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
            0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
            0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
            0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
            0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
            0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
            0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
            0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
            0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
            0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
            0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
            0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
            0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
        ]

        # Dữ liệu ban đầu
        self.data = string

        # Dữ liệu chuẩn bị cho quá trình băm
        self.w = []

        self.encrypt_message = None

    # Function chuyển dữ liệu ban đầu thành các chuỗi nhị phân
    def prepare_data(self):
        result = Converter.string_to_binary(self.data) + '10000000'
        length = len(result)
        result = result.ljust((length + 64 + 511) // 512 * 512 - 64, '0')
        result += bin(len(self.data.encode('utf-8')) * 8)[2:].zfill(64)
        self.w = [result[32 * i: 32 * (i + 1)] for i in range(16)] + ['0' *
32] * 48
```

```
    # Function xử lý dữ liệu
    def data_processing(self):
        for i in range(16, 64):
            s0 = Converter.xor_string(Converter.right_rotate(self.w[i - 15],
7), Converter.right_rotate(self.w[i - 15], 18),
Converter.right_shift(self.w[i - 15], 3))
            s1 = Converter.xor_string(Converter.right_rotate(self.w[i - 2],
17), Converter.right_rotate(self.w[i - 2], 19),
Converter.right_shift(self.w[i - 2], 10))
            self.w[i] = Converter.add_binary_string(self.w[i - 16], s0,
self.w[i - 7], s1)

    # Function chi tiết từng vòng băm
    def detail_encrypt(self, number):
        s1 = Converter.xor_string(Converter.right_rotate(self.index[4], 6),
Converter.right_rotate(self.index[4], 11),
Converter.right_rotate(self.index[4], 25))
        choice = Converter.xor_string(Converter.and_string(self.index[4],
self.index[5]), Converter.and_string(Converter.not_string(self.index[4]),
self.index[6]))
        temp1 = Converter.add_binary_string(self.index[7], s1, choice,
self.k_values[number], self.w[number])
        s0 = Converter.xor_string(Converter.right_rotate(self.index[0], 2),
Converter.right_rotate(self.index[0], 13),
Converter.right_rotate(self.index[0], 22))
        majority = Converter.xor_string(Converter.and_string(self.index[0],
self.index[1]), Converter.and_string(self.index[0], self.index[2]),
Converter.and_string(self.index[1], self.index[2]))
        temp2 = Converter.add_binary_string(s0, majority)
        self.index[7] = self.index[6]
        self.index[6] = self.index[5]
        self.index[5] = self.index[4]
        self.index[4] = Converter.add_binary_string(self.index[3], temp1)
        self.index[3] = self.index[2]
        self.index[2] = self.index[1]
        self.index[1] = self.index[0]
        self.index[0] = Converter.add_binary_string(temp1, temp2)
        result = ""
        for i in range(8):
```

```
            result += self.index[i]
        self.all_data.append(result)

    # Function biểu diễn băm dữ liệu
    def encrypt(self):
        self.prepare_data()
        self.data_processing()
        self.const = [bin(int(x, 16))[2:].zfill(32) for x in self.const]
        self.k_values = [bin(x)[2:].zfill(32) for x in self.k_values]
        for i in range(8):
            self.index.append(self.const[i])
        for i in range(64):
            self.detail_encrypt(i)
        for i in range (8):
            self.const[i] = Converter.add_binary_string(self.const[i],
self.index[i])
        self.encrypt_message = ''.join(Converter.binary_to_hex(x) for x in
self.const)

    # Function băm dữ liệu bằng thư viện có sẵn
    @staticmethod
    def encrypt_hashlib(string):
        sha256_hash = hashlib.sha256()
        sha256_hash.update(string.encode())
        return sha256_hash.hexdigest()

    # Function so sánh thời gian giữa không sử dụng và sử dụng thư viện
    @staticmethod
    def time_running(string):
        start_time = time.time()
        sha256_obj = SHA256Hash(string)
        sha256_obj.encrypt()
        end_time = time.time()
        time1 = end_time - start_time
        start_time = time.time()
        SHA256Hash.encrypt_hashlib(string)
        end_time = time.time()
        time2 = end_time - start_time
        return time1, time2
```

Lớp SHA256App

```
class SHA256App(SHA256Hash):
    def __init__ (self, root_index, string):
        super().__init__(string)
        self.root = root_index
        self.data = string

        self.root.title("Chi tiết hàm băm SHA-256")
        button_frame = tk.Frame(self.root)
        button_frame.pack(pady = 10)

        self.init_button = tk.Button(button_frame, text = "Dữ liệu ban đầu",
command = self.first_data)
        self.init_button.grid(row = 0, column = 0, padx = 5)

        self.detail_button = tk.Button(button_frame, text="Chuẩn bị dữ
liệu", command=self.detail_data)
        self.detail_button.grid(row=0, column=1, padx = 5)

        self.process_button = tk.Button(button_frame, text = "Xử lý dữ
liệu", command = self.process_data)
        self.process_button.grid(row = 0, column = 2, padx = 5)

        self.step_button = tk.Label(button_frame, text = "Kết quả của bước")
        self.step_button.grid(row = 0, column = 3, padx = 5)

        # Chọn vòng băm
        self.step_number = ttk.Combobox(button_frame, values = list(range(1,
65)), width = 5)
        self.step_number.grid(row = 0, column = 4, padx = 5)
        self.step_number.bind("<<ComboboxSelected>>", self.get_step_number)

        self.final_button = tk.Button(button_frame, text = "Dữ liệu cuối
cùng", command = self.final_data)
        self.final_button.grid(row = 0, column = 5, padx = 5)

        self.encrypt_button = tk.Button(button_frame, text = "Kết quả mã
hóa", command = self.encrypt_data)
        self.encrypt_button.grid(row = 0, column = 6, padx = 5)
```

```
        self.save_button = tk.Button(button_frame, text = "Lưu kết quả mã
hóa", command = self.save_data)
        self.save_button.grid(row = 0, column = 7, padx = 5)
        # Vùng để hiển thị kết quả từng bước
        self.output_text = tk.Text(root_index, height = 10, width = 100)
        self.output_text.pack(pady = 10)

    def first_data(self):
        result = Converter.string_to_binary(self.data) + '10000000'
        length = len(result)
        result = result.ljust((length + 64 + 511) // 512 * 512 - 64, '0')
        result += bin(len(self.data.encode('utf-8')) * 8)[2:].zfill(64)
        data_binary = [result[32 * i: 32 * (i + 1)] for i in range(16)]
        temp = ""
        for i in range(16):
            temp += f"{data_binary[i]}\n"
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, temp)

    def detail_data(self):
        self.prepare_data()
        result = ""
        for i in range(64):
            result += f"w[{i}]: {self.w[i]}\n"
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    def process_data(self):
        self.prepare_data()
        self.data_processing()
        result = ""
        for i in range(64):
            result += f"w[{i}]: {self.w[i]}\n"
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    def encrypt_data(self):
        sha256_obj = SHA256Hash(self.data)
        sha256_obj.encrypt()
```

```
        hashed_message_library = SHA256Hash.encrypt_hashlib(self.data)
        time1, time2 = SHA256Hash.time_running(self.data)
        str1 = f"Kết quả mã hóa: \n"
        str2 = f"Không sử dụng thư viện: {sha256_obj.encrypt_message}\nThời
gian thực hiện: {time1} giây\n"
        str3 = f"Sử dụng thư viện: {hashed_message_library}\nThời gian thực
hiện: {time2} giây"
        result = str1 + str2 + str3
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    # Function hiển thị kết quả của từng vòng băm
    def get_step_number(self, event):
        number = int(self.step_number.get())
        result = ""
        sha256_hash = SHA256Hash(self.data)
        sha256_hash.encrypt()
        index = sha256_hash.all_data[number - 1]
        temp = [index[32*i: 32*(i+1)] for i in range(8)]
        for i in range(8):
                result += f"index[{i}]: {temp[i]}\n"
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    def final_data(self):
        sha256_variable = SHA256Hash(self.data)
        sha256_variable.prepare_data()
        sha256_variable.data_processing()
        sha256_variable.const = [bin(int(x, 16))[2:].zfill(32) for x in
sha256_variable.const]
        sha256_variable.k_values = [bin(x)[2:].zfill(32) for x in
sha256_variable.k_values]
        for i in range(8):
            sha256_variable.index.append(sha256_variable.const[i])
        for i in range(64):
            sha256_variable.detail_encrypt(i)
        for i in range(8):
            sha256_variable.const[i] =
Converter.add_binary_string(sha256_variable.const[i],
sha256_variable.index[i])
```

```
        result = ""
        for i in range(8):
            result += f"const[{i}]: {sha256_variable.const[i]} -> Chuyển
sang hệ 16: {Converter.binary_to_hex(sha256_variable.const[i])}\n"
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    def save_data(self):
        file_path = filedialog.asksaveasfilename(
            title = "Lưu file",
            defaultextension = ".txt",
            filetypes = [("Text Files", "*.txt"), ("All Files", "*.*")]
        )
        sha256_obj = SHA256Hash(self.data)
        sha256_obj.encrypt()
        if file_path:  # Nếu người dùng chọn file
            try:
                with open(file_path, 'w', encoding='utf-8') as file:
                    file.write(sha256_obj.encrypt_message)
                print(f"Dữ liệu đã được lưu vào {file_path}")
            except Exception as e:
                print(f"Lỗi khi lưu file: {e}")
        else:
            print("Hủy lưu file.")
```

Lớp NumericTool

```
class NumericTool:
    # Function tìm ước chung lớn nhất của 2 số nguyên bằng thuật toán
Euclid
    @staticmethod
    def euclid_algorithm(a, b):
        while b != 0:
            a, b = b, a % b
        return a

    # Function hiển thị quá trình tìm ước
    @staticmethod
    def euclid_gui(a, b):
        result = [["Lần lặp", "e", "n", "Số dư"]]
        i = 0
```

```
        while b != 0:
            r = a % b
            result.append([f"Lần {i + 1}", a, b, r])
            a, b = b, r
            i += 1
        result.append([f"Lần {i+1}", a, b, 0])
        return result

    # Function hiển thị quá trình tìm phần tử nghịch đảo
    @staticmethod
    def extended_euclid_gui(a, b):
        result = [["Lần lặp", "A3", "B3", "R3", "A2", "B2", "R2"]]
        a1, a2, a3 = 1, 0, a
        b1, b2, b3 = 0, 1, b
        i = 0
        while b3 != 0 and b3 != 1:
            q = a3 // b3
            r1 = a1 - q*b1
            r2 = a2 - q*b2
            r3 = a3 - q*b3
            result.append([f"Lần {i + 1}", a3, b3, r3, a2, b2, r2])
            a1, a2, a3 = b1, b2, b3
            b1, b2, b3 = r1, r2, r3
            i += 1
        return result

    # Function tìm phần tử nghịch đảo của b modulo a bằng thuật toán Euclid
mở rộng
    @staticmethod
    def extended_euclid(a, b):
        a1, a2, a3 = 1, 0, a
        b1, b2, b3 = 0, 1, b
        while b3 != 0 and b3 != 1:
            q = a3 // b3
            r1 = a1 - q * b1
            r2 = a2 - q * b2
            r3 = a3 - q * b3
            a1, a2, a3 = b1, b2, b3
            b1, b2, b3 = r1, r2, r3
        if b3 == 0:
```

```
            return a3
        if b3 == 1:
            return b2

    # Function tạo số nguyên lớn
    @staticmethod
    def generate_prime_candidate(length = 1024):
        p = random.getrandbits(length)
        p |= (1 << length - 1) | 1
        return p

    # Function tạo số nguyên tố lớn
    @staticmethod
    def generate_prime_number(length = 1024):
        p = 4
        while not isprime(p):
            p = NumericTool.generate_prime_candidate(length)
        return p
```

Lớp RSAEncryptionApp

```
class RSAEncryptionApp:
    def __init__(self, root_index, string):
        self.root = root_index
        self.data = string

        self.root.title("Chi tiết mã hóa RSA")
        button_frame = tk.Frame(self.root)
        button_frame.pack(pady = 2)

        # Button dùng để nhập độ dài khóa với độ dài mặc định là 1024 bits
        self.length_button = tk.Label(button_frame, text = "Nhập độ dài
khóa")
        self.length_button.grid(row = 0, column = 0, padx = 5)
        self.length = tk.Entry(button_frame, width = 10)
        self.length.grid(row = 0, column = 1, padx = 5)

        self.prime_number_button = tk.Button(button_frame, text = "Sinh cặp
số nguyên tố", command = self.generate_keypair)
        self.prime_number_button.grid(row = 0, column = 2, padx = 5)
```

```
        self.gui_button = tk.Button(button_frame, text = "Kiểm tra hai số
nguyên tố cùng nhau", command = self.gui_euclid)
        self.gui_button.grid(row = 0, column = 3, padx = 5)

        self.inverse_button = tk.Button(button_frame, text = "Tìm phần tử
nghịch đảo", command = self.gui_inverse)
        self.inverse_button.grid(row = 0, column = 4, padx = 5)

        self.key_button = tk.Button(button_frame, text = "Hiển thị cặp
khóa", command = self.display_key)
        self.key_button.grid(row = 0, column = 5, padx = 5)

        self.key_save = tk.Button(button_frame, text = "Lưu khóa", command
= self.save_key)
        self.key_save.grid(row = 0, column = 6, padx = 5)

        self.encrypt_button = tk.Button(button_frame, text = "Mã hóa",
command = self.encrypt)
        self.encrypt_button.grid(row = 0, column = 7, padx = 5)

        self.save_button = tk.Button(button_frame, text = "Lưu kết quả mã
hóa", command = self.save_data)
        self.save_button.grid(row = 0, column = 8, padx = 5)

        self.decrypt_button = tk.Button(button_frame, text = "Giải mã",
command = self.decrypt)
        self.decrypt_button.grid(row = 0, column = 9, padx = 5)
        self.p = None
        self.q = None
        self.N = None
        self.n = None
        self.e = None
        self.g = None
        self.d = None
        self.encrypt_message = None
        self.decrypt_message = None
        self.output_text = tk.Text(root_index, height = 10, width = 150)
        self.output_text.pack(pady = 10)

    def generate_keypair(self):
```

```
        content = self.length.get()
        length = 1024
        if content.strip():
            length = int(content)
        self.p = NumericTool.generate_prime_number(length)
        self.q = NumericTool.generate_prime_number(length)
        self.N = self.p * self.q
        self.n = (self.p - 1) * (self.q - 1)
        result = f"Cặp số nguyên tố: ({self.p}, {self.q})\n -> Chuyển sang
hệ 16: ({hex(self.p)[2:]}, {hex(self.q)[2:]})\n"
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    def gui_euclid(self):
        column_width = [20, 20, 20, 20]
        self.output_text.delete(1.0, tk.END)
        self.e = random.randrange(1, self.n)
        self.g = NumericTool.euclid_algorithm(self.e, self.n)
        while self.g != 1:
            self.e = random.randrange(1, self.n)
            self.g = NumericTool.euclid_algorithm(self.e, self.n)
        result = NumericTool.euclid_gui(self.e, self.n)
        headers = result[0]
        header_row = "".join(f"{str(headers[i]).ljust(column_width[i])}"
for i in range(len(headers)))
        self.output_text.insert("end", header_row + "\n")
        for row in result[1:]:
            row_text = "".join(f"{str(row[i]).ljust(column_width[i])}" for
i in range(len(row)))
            self.output_text.insert("end", row_text + "\n")
        self.output_text.insert("end", f"-> Ước chung lớn nhất là {self.g}
nên {self.e} và {self.n} là hai số nguyên tố cùng nhau")

    def gui_inverse(self):
        column_width = [15, 15, 15, 15, 15, 15, 15]
        self.output_text.delete(1.0, tk.END)
        result = NumericTool.extended_euclid_gui(self.n, self.e)
        temp = NumericTool.extended_euclid(self.n, self.e)
        if temp >= 0:
            self.d = temp
```

```
        else:
            self.d = self.n + temp
        headers = result[0]
        header_row = "".join(f"{str(headers[i]).ljust(column_width[i])}"
for i in range(len(headers)))
        self.output_text.insert("end", header_row + "\n")
        for row in result[1:]:
            row_text = "".join(f"{str(row[i]).ljust(column_width[i])}" for
i in range(len(row)))
            self.output_text.insert("end", row_text + "\n")
        self.output_text.insert("end", f"-> Phần tử nghịch đảo của {self.e}
trong modulo {self.n} là {self.d}, sử dụng thư viện là {mod_inverse(self.e,
self.n)}")

    def display_key(self):
        public_key = f"Khóa công khai: ({self.e}, {self.n})\n"
        private_key = f"Khóa bí mật: ({self.d}, {self.n})\n"
        result = public_key + private_key
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    def encrypt(self):
        start_time = time.time()
        self.encrypt_message = [hex(pow(ord(char), self.d, self.N))[2:] for
char in self.data]
        end_time = time.time()
        time_running = end_time - start_time
        result = f"Kết quả mã hóa: {''.join(self.encrypt_message)}\nThời
gian mã hóa: {time_running} giây"
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    def decrypt(self):
        start_time = time.time()
        file_data_path = filedialog.askopenfilename()
        if file_data_path:
            with open(file_data_path, 'r', encoding = 'utf-8') as file:
                self.encrypt_message = [line.strip() for line in file]
        file_key_path = filedialog.askopenfilename()
        if file_key_path:
```

```
            with open(file_key_path, 'r', encoding = 'utf-8') as file:
                self.e = int(file.readline().strip())
                self.N = int(file.readline().strip())
        self.decrypt_message = [chr(pow(int(char, 16), self.e, self.N)) for
char in self.encrypt_message]
        end_time = time.time()
        time_running = end_time - start_time
        result = f"Kết quả giải mã: {''.join(self.decrypt_message)}\nThời
gian giải mã: {time_running} giây"
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, result)

    def save_data(self):
        file_path = filedialog.asksaveasfilename(
            title = "Lưu file",
            defaultextension = ".txt",
            filetypes = [("Text Files", "*.txt"), ("All Files", "*.*")]
        )
        if file_path:  # Nếu người dùng chọn file
            try:
                with open(file_path, 'w', encoding='utf-8') as file:
                    for item in self.encrypt_message:
                        file.write(f"{item}\n")
                print(f"Dữ liệu đã được lưu vào {file_path}")
            except Exception as e:
                print(f"Lỗi khi lưu file: {e}")
        else:
            print("Hủy lưu file.")

    def save_key(self):
        file_path = filedialog.asksaveasfilename(
            title = "Lưu file",
            defaultextension = ".txt",
            filetypes = [("Text Files", "*.txt"), ("All Files", "*.*")]
        )
        if file_path:  # Nếu người dùng chọn file
            try:
                with open(file_path, 'w', encoding='utf-8') as file:
                    file.write(f"{self.e}\n{self.N}")
                print(f"Khóa đã được lưu vào {file_path}")
```

```
            except Exception as e:
                print(f"Lỗi khi lưu file: {e}")
        else:
            print("Hủy lưu file.")
```

Chương trình chính

```
if __name__ == "__main__":
    root = tk.Tk()
    app = EncryptApp(root)
    root.mainloop()
```