ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

NGUYỄN DƯƠNG MINH TÂM **MSHV: 22C12005**

PHAN LẠI NHẬT MINH **MSHV: 22C12004**

LÊ TRƯỜNG VĨ **MSHV: 22C12007**

MÃ HÓA RSA

&

CHƯƠNG TRÌNH TÍNH TOÁN RSA

Đồ án nghiên cứu

Chương trình cao học và nghiên cứu sinh

Môn: Phương Pháp toán tin học và giải thuật

TP. HỒ CHÍ MINH – 2023

MỤC LỤC

PHẦN MỞ ĐẦU

Mật mã học là một lĩnh vực liên quan đến các kỹ thuật ngôn ngữ và toán học để đảm bảo an toàn thông tin, cụ thể là trong thông tin liên lạc. Trong lịch sử, mật mã học gắn liền với quá trình mã hóa; điều này có nghĩa là nó gắn với các cách thức để chuyển đổi thông tin từ dạng này sang dạng khác hay nói cách khác làm cho thông tin trở thành dạng không thể đọc được nếu như không có các thông tin bí mật. Quá trình mã hóa được sử dụng chủ yếu để đảm bảo tính bảo mật thông tin, chẳng hạn trong công tác tình báo, quân sự hay ngoại giao cũng như các bí mật về kinh tế, thương mại.

Ví dụ, ta muốn trao đổi thông tin bí mật nào đó với đối tác, nếu gặp mặt trực tiếp chỉ cần ngồi cạnh và nói nhỏ với người đó. Tuy nhiên, không phải lúc nào đối tác ấy cũng bên cạnh, ta chỉ còn cách gọi điện, gửi mail,... Những cách này sẽ không thật sự an toàn, nếu ai đó có động cơ hay ý đồ muốn xâm nhập chiếm lấy thông tin của tôi. Điều này khiến cho kẻ muốn đánh cắp thông tin dễ dàng thực hiện hành vi như đặt nghe lén ghi âm điện thoại, ăn trộm,…Và một trong những giải pháp ngăn chặn có kẻ xâm nhập vào thông tin mật là mã hoá nó. Có nghĩa là ta sẽ thêm 1 mã hóa phức nào đó vào thông tin của mình, và chỉ có những người có quyền truy cập vào thông tin này và mới có thể đọc và chia sẻ thông tin.

Những năm gần đây, lĩnh vực hoạt động của mật mã hóa đã được mở rộng: mật mã hóa hiện đại cung cấp cơ chế cho nhiều hoạt động hơn là chỉ duy nhất việc giữ bí mật và có một loạt các ứng dụng như: chứng thực khóa công khai, chữ ký số, bầu cử điện tử hay tiền điện tử,..

CHƯƠNG 1

MÃ HÓA RSA

1.1. Mã hóa đối xứng và mã hóa bất đối xứng

Hệ mã hóa đối xứng đề cập đến việc sử dụng cùng một khóa cho việc mã hóa và giải mã. Thông tin sẽ được mã hóa theo một phương pháp nhất định tương ứng với một khóa. Khóa này cần phải được giữ bí mật, chỉ có người tạo mã và người nhận biết được, nếu khóa bị lộ thì người ngoài sẽ dễ dàng giải mã và có được thông tin.

Hệ mã hóa bất đối xứng (mã hóa công khai) sử dụng một cặp khóa để mã hóa và giải mã, trong đó:

- Khóa công khai (public key) dùng để mã hóa thông tin cần chia sẻ.
- Khóa bí mật (private key) dùng để giải mã. Khóa này thuộc sở hữu riêng tư của của chủ sở hữu, không phân phát hay chia sẻ cho bất cứ ai.

Điều này có nghĩa là, bất cứ ai cũng có thể sử dụng khóa công khai để mã hóa bản tin và gửi cho người nhận. Nhưng chỉ có người nhận sở hữu khóa bí mật mới có thể giải mã và đọc các thông điệp được gửi. Bất cứ ai không có khóa bí mật thì không thể giải mã và đọc thông tin

Nhìn chung thì công thức để mã hóa và giải mã các thông điệp tin nhắn là như nhau. Tất cả các tin nhắn được mã hóa bằng khóa công khai chỉ có thể được giải mã bằng khóa riêng tư và các tin nhắn được mã hóa bằng khóa riêng tư chỉ có thể được giải mã bằng khóa công khai. Tuy nhiên việc mã hóa tin nhắn bằng khóa riêng tư này lại không thực sự có ý nghĩa, vì khóa công khai được mọi người biết đến, bất kỳ ai cũng có thể giải mã được. tuy nhiên, mã hóa bằng khóa công khai được sử dụng theo cách khác là để chứng minh rằng tin nhắn thực sự đến từ chủ sở hữu của khóa riêng tư. Đó chính là ứng dụng cho chữ ký số. Người chủ dùng khoá riêng tư để ký tên và các người khác dùng khóa công khai để xác minh chữ ký đó có phải chính chủ không.

Các thuật toán mã khóa công khai là thành phần bảo mật cơ bản trong các hệ thống mật. Chẳng hạn như trong Transport Layer Security (TLS), S/MIME, PGP hay GPG. Một số thuật toán mã khóa công khai cung cấp tính bảo mật và phân phối khóa (ví dụ: trao đổi khóa Diffie–Hellman) hay chữ ký số (ví dụ: Thuật toán chữ ký số) và một số cung cấp cả hai (ví dụ: RSA).

1.2. Thuật toán RSA

RSA (Rivest–Shamir–Adleman) là một trong những hệ thống mật mã khóa công khai đầu tiên và được sử dụng rộng rãi trong việc truyền dữ liệu an toàn. Từ viết tắt RSA là các chữ cái đầu trong họ của Ron Rivest, Adi Shamir và Leonard Adleman, những người đã mô tả công khai thuật toán này vào năm 1977.

Trong 1 hệ thống mật mã, khóa mã hóa là khóa công khai và khóa giải mã được giữ bí mật (riêng tư). Người dùng thuật toán RSA dựa trên hai số nguyên tố lớn và giá trị phụ trợ mà tạo khóa công khai và sau đó công khai khóa. Các số nguyên tố phải được giữ bí mật. Bất kỳ ai cũng có thể sử dụng khóa công khai để mã hóa thông điệp được gửi đi, nhưng chỉ người có biết về cặp số nguyên tố mới có thể giải mã tin nhắn.

Tuy nhiên RSA vẫn có thể bị phá và được gọi là sự cố RSA. Nhưng ở thời điểm hiện tại, không có phương pháp nào có thể đánh bại hệ thống nếu hệ thống sử dụng khóa đủ lớn.

Về lý thuyết thuật toán RSA phát biểu như sau:

Cho p và q là 2 số nguyên tố khác nhau, $n = pq,\ \varphi(n) = (p-1)(q-1)$ *và e,d là hai số nguyên sao cho* $ed \equiv 1(mod\ \varphi(n))$*.Thì với tất cả* $m \in\ Z_n$ *chúng ta có* $m^{ed} \equiv m(mod\ n)$

Như vậy việc mã hóa và giải mã dựa trên lý thuyết RSA như sau: mật mã sẽ được tạo trên tập Z_n, một văn bản thô $m \in Z_n$ sẽ được mã hóa thành c với cặp khóa công khai {e, n} bằng công thức $m^e \equiv c(mod\ n)$ và c sẽ được giải mã với cặp khóa bí mật {d, n}bằng công thức $c^d \equiv m(mod\ n)$

1.2.1. Chuyển đổi văn bản rõ

Trước khi thực hiện mã hóa, ta phải thực hiện việc chuyển đổi văn bản rõ (chuyển đổi từ M sang m) sao cho không có giá trị nào của M tạo ra văn bản mã không an toàn. Nếu không có quá trình này, RSA sẽ gặp phải một số vấn đề sau:

- Nếu m = 0 hoặc m = 1 sẽ tạo ra các bản mã có giá trị là 0 và 1 tương ứng
- Khi mã hóa với số mũ nhỏ (chẳng hạn e = 3) và m cũng có giá trị nhỏ, giá trị m^e cũng nhận giá trị nhỏ (so với n). Như vậy phép môđun không có tác dụng và có thể dễ dàng tìm được m bằng cách khai căn bậc e của c (bỏ qua môđun).
- RSA là phương pháp mã hóa xác định (không có thành phần ngẫu nhiên) nên kẻ tấn công có thể thực hiện tấn công lựa chọn bản rõ bằng cách tạo ra một bảng tra giữa bản rõ và bản mã. Khi gặp một bản mã, kẻ tấn công sử dụng bảng tra để tìm ra bản rõ tương ứng.

Trên thực tế, ta thường gặp 2 vấn đề đầu khi gửi các bản tin ASCII ngắn với m là nhóm vài ký tự ASCII. Một đoạn tin chỉ có 1 ký tự NULL sẽ được gán giá trị m = 0 và cho ra bản mã là 0 bất kể giá trị của e và N. Tương tự, một ký tự ASCII khác, SOH, có giá trị 1 sẽ luôn cho ra bản mã là 1. Với các hệ thống dùng giá trị e nhỏ thì tất cả ký tự ASCII đều cho kết quả mã hóa không an toàn vì giá trị lớn nhất của m chỉ là 255 và 2553 nhỏ hơn giá trị n chấp nhận được. Những bản mã này sẽ dễ dàng bị phá mã.

Để tránh gặp phải những vấn đề trên, RSA trên thực tế thường bao gồm một hình thức chuyển đổi ngẫu nhiên hóa m trước khi mã hóa. Quá trình chuyển đổi này phải đảm bảo rằng m không rơi vào các giá trị không an toàn. Sau khi chuyển đổi, mỗi bản rõ khi mã hóa sẽ cho ra một trong số khả năng trong tập hợp bản mã. Điều này làm giảm tính khả thi của phương pháp tấn công lựa chọn bản rõ (một bản rõ sẽ có thể tương ứng với nhiều bản mã tuỳ thuộc vào cách chuyển đổi).

Một số tiêu chuẩn, chẳng hạn như PKCS, đã được thiết kế để chuyển đổi bản rõ trước khi mã hóa bằng RSA. Các phương pháp chuyển đổi này bổ sung thêm bít vào M. Các phương pháp chuyển đổi cần được thiết kế cẩn thận để tránh những dạng tấn công phức tạp tận dụng khả năng biết trước được cấu trúc của bản rõ. Phiên bản ban đầu của PKCS dùng một phương pháp đặc ứng (ad-hoc) mà về sau được biết là không an toàn trước tấn công lựa chọn bản rõ thích ứng (adaptive chosen ciphertext attack). Các phương pháp chuyển đổi hiện đại sử dụng các kỹ thuật như chuyển đổi mã hóa bất đối xứng tối ưu (Optimal Asymmetric Encryption Padding - OAEP) để chống lại tấn công dạng này. Tiêu chuẩn PKCS còn được bổ sung các tính năng khác để đảm bảo an toàn cho chữ ký RSA (Probabilistic Signature Scheme for RSA - RSA-PSS).

1.2.2. Khởi tạo số nguyên tố lớn

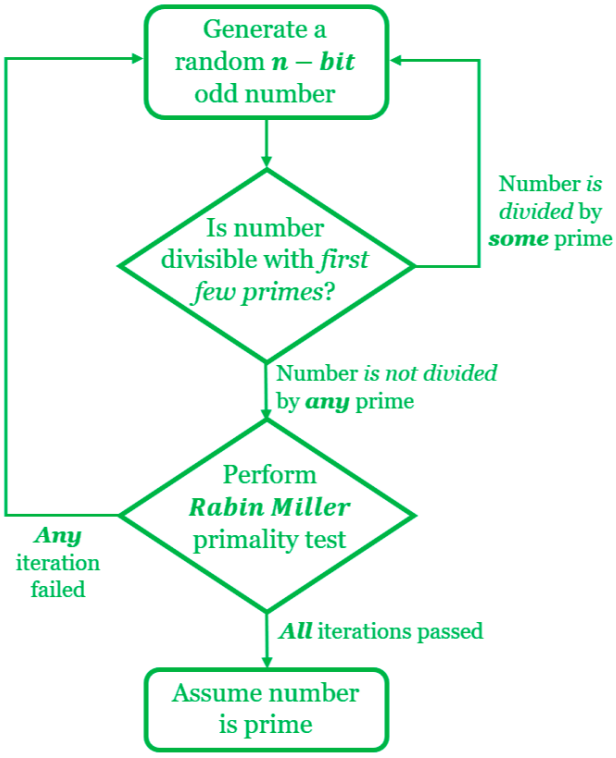
Tính bảo mật của thuật toán RSA dựa trên độ khó của việc phân tích các số rất lớn. Việc thiết lập hệ thống mật mã RSA liên quan đến việc tạo ra hai số nguyên tố lớn, chẳng hạn như p và q, từ đó, mô đun RSA được tính là $n = p * q$. Kích thước mô-đun càng lớn thì mức độ bảo mật của hệ thống RSA càng cao. Kích thước mô-đun RSA được khuyến nghị là 2048 bit đến 4096 bit.

Để tổng hợp các số nguyên tố lớn như thế, ta dựa vào sàng Eratosthenes và phương pháp kiểm tra tính nguyên tố Rabin Miller. Mục đích để tìm các số nguyên tố với mức xác suất cao thỏa đáng.

Các bước tính toán hiệu quả các số nguyên tố ngẫu nhiên rất lớn với kích thước bit được chỉ định.

- Chọn trước một số ngẫu nhiên với kích thước bit mong muốn
- Đảm bảo số đã chọn không chia hết cho vài trăm số nguyên tố đầu tiên (những số này được tạo trước, sử dụng sàng Eratosthenes để tính toán)

- Áp dụng một số lần lặp trong việc kiểm tra tính nguyên tố Rabin Miller, dựa trên tỷ lệ lỗi chấp nhận được, để nhận được một số có thể là số nguyên tố



Sàng Eratosthenes

Sàng Eratosthenes dùng để tìm các số nguyên tố nhỏ hơn hoặc bằng số nguyên *N* nào đó. Nó còn có thể được sử dụng để kiểm tra một số nguyên nhỏ hơn hoặc bằng *N* hay không.

Nguyên lý hoạt động của sàng là vào mỗi lần duyệt, ta chọn một số nguyên tố và loại ra khỏi sàng tất cả các bội của số nguyên tố đó mà lớn hơn số đó. Sau khi duyệt xong, các số còn lại trong sàng đều là số nguyên tố.

Kiểm tra tính nguyên tố bằng Rabin Miller.

Kiểm tra Miller-Rabin là một thuật toán xác suất để kiểm tra tính nguyên tố. Khi sử dụng kiểm tra Miller-Rabin chúng ta căn cứ vào một mệnh đề *Q(p, a)* đúng

với các số nguyên tố p và mọi số tự nhiên $a \in A \subset N$ và kiểm tra xem chúng có đúng với số *n* muốn kiểm tra và một số $a \in A$ được chọn ngẫu nhiên hay không. Nếu mệnh đề *Q(n, a)* không đúng, thì n không phải là số nguyên tố, còn nếu *Q(n, a)* đúng, số *n* có thể là số nguyên tố với một xác suất nào đó. Khi tăng số lần thử, xác suất để *n* là số nguyên tố tăng lên.

1.2.3. Khởi tạo khóa công khai và khóa riêng tư

Với thuật toán RSA bước đầu tiên để tạo hệ mã là cần tạo ra cho mình cặp khóa công khai và khóa bí mật. Như đã đề cặp mã hóa RSA dựa trên độ khó của bài toán phân tích ra thừa số nguyên tố đặc biệt với số nguyên lớn. Do đó, độ mạnh của mã hóa hoàn toàn phụ thuộc vào kích thước khóa và nếu chúng ta tăng gấp đôi hoặc gấp ba kích thước khóa, độ mạnh của mã hóa sẽ tăng theo cấp số nhân.

Cho ví dụ minh họa như sau: Giả sử An và Bình cần trao đổi thông tin bí mật thông qua một kênh không an toàn (ví dụ như Messenger).Với thuật toán RSA, An đầu tiên cần tạo ra cho mình cặp khóa gồm khóa công khai và khóa bí mật theo các bước sau:

1/ Chọn 2 số nguyên tố lớn khác nhau *p* và *q* ngẫu nhiên và độc lập.

2/ Tính $n = p * q$

3/ Tính giá trị hàm số Euler: $\varphi(n) = (p - 1)(q - 1)$

4/ Chọn một số *e* với $1 < e < \varphi(n)$ và là số nguyên tố cùng nhau với $\varphi(n)$.

5/ Tính *d* sao cho $de \equiv 1\ (mod\ \varphi(n))$, (*d* là nghịch đảo nhân của e trên $\varphi(n)$)

Như vậy khóa công khai bao gồm *{e, n}* và khóa bí mật là *{d, n}.* An sẽ gửi khóa công khai cho Bình và giữ bí mật khóa cá nhân của mình.

Lưu ý: Các bước trên được thực hiện bằng *giải thuật Euclid* trong Modun số học

1.2.4. Thuật toán mã hóa & giải mã

Mã hóa bằng khóa công khai

Trong bước này chúng ta chuyển đổi văn bản thô thành dang $m \in Z_n$ bằng cách chuyển đổi chuỗi char thành chuỗi số là thứ tự của char trong bảng Unicode.

Sau đó dùng cặp khóa công khai $\{e, n\}$ để mã hóa chuỗi m thành mật mã c bằng công thức:

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

Bình sao khi dùng cặp khóa công khai để mã hóa tin nhắn m thành mật mã c, và sẽ gửi sẽ gửi c cho An người duy nhất có khóa bí mật $\{d, n\}$ mới có thể giải mã.

Giải mã bằng khóa bí mật

An sau khi nhận được mật mã c sẽ dung khóa bí mật của mình để giải mã c thành tin nhắn m bằng công thức:

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

Biết m, An tìm lại được thông tin văn bản m ban đầu theo phương pháp đã thỏa thuận trước.

1.2.5. Thuật toán nhân lũy thừa bằng bình phương và ví dụ minh họa

Thuật toán nhân lũy thừa bằng bình phương

Các phép toán trong mã hóa và giải mã đều có thể thực hiện hiệu quả bằng phương pháp tính hàm mũ (theo môđun) bằng (thuật toán bình phương và nhân, là thuật toán tính nhanh lũy thừa tự nhiên của một số, trong trường hợp cơ số là số nguyên có thể rút gọn theo một Modun.

Xét $x^n = x * x * \ldots * x \ \ (n \ lần)$

Ta phân tích ví dụ:

$x^2 = x * x$

$x^3 = x^2 *$x

Như vậy ta có công thức đệ quy:

1/ Với n=0 thì $x^n = 1$

2/ Với n >0 ta có công thức:

$$f(n) = \{(x^k)^2, khi\ n = 2k\ (x^k)^2 * x\ \ , khi\ n = 2k + 1\}$$

Sau đây là ví dụ với số cụ thể, cho *p = 7, q =17, e = 11*. Chúng ta thực hiện các phép để tính cặp khóa cá nhân và công khai. Mã hóa thông điệp *m = 20* và sau đó giải mã.

Đầu tiên ta có

n = p × q = 7 × 17 = 119

φ(n) = (7 – 1) × (17 – 1) = 96

Tính $d = e^{-1}\ (mod\ \varphi(n)\ =\ 11^{-1}\ (mod\ 96)\ = 35$

Vậy khóa công khai: *{e, n}* = {11, 119}, khóa bí mật*: {d, n}* = {35,119}

- Mã hóa *m = 20*

Mật mã $c = m^e = 20^{11}$ = 41 bằng thuật toán bình phương và nhân như sau:

1011 (e =11)	m mod n	1(mod n=199)
1	20^2 = 400 = 43	20× 1 = 20
1	$43^2 = 1849 = 64$	20× 43 = 860 = 27
0	$64^2 = 4096 = 50$	
1	_	27 ×50 =1350=41

- Giải mã c bằng khóa bí mật *d = 35*

m = $c^d = 41^{35} = 20$ (đúng với giá trị tin nhắn ban đầu)

100011 (d=35)	41	1
1	$41^2 = 1681 = 15$	$1\times 41 = 41$
1	$15^2 = 225 = -13$	$41\times 15 = 615 = 20$
0	$-13^2 = 169 = 50$	
0	$50^2 = 2500 = 1$	
0	$1^2 = 1$	
1		$20\times 1 = 20$

1.2.6. Chữ ký số

Thuật toán RSA còn được dùng để tạo chữ ký số cho văn bản. Giả sử An muốn gửi cho Bình một văn bản có chữ ký của mình. Để làm việc này, An tạo ra một giá trị băm (hash value) của văn bản cần ký và tính giá trị mũ d mod n của giá trị băm , $chữ\ ký\ = hashV^d$. Giá trị cuối cùng chính là chữ ký điện tử của văn bản đang xét.

Khi Bình nhận được văn bản cùng với chữ ký điện tử, anh ta tính giá trị mũ e mod n của chữ ký($hashV = chữ\ ký^e$) đồng thời với việc tính giá trị băm của văn bản. Nếu 2 giá trị này như nhau thì Bình biết rằng người tạo ra chữ ký biết khóa bí mật của An và văn bản đã không bị thay đổi sau khi ký.

CHƯƠNG 2

GIẢI THÍCH MÃ NGUỒN THUẬT TOÁN MÃ HÓA RSA

Chương trình mô tả thuật toán mã hóa RSA này được viết bằng ngôn ngữ Python, và sử dụng thư viện hỗ trợ *random* và *math.*

Những tác vụ chính của chương trình bao gồm:

- Chuyển đổi, chuẩn hóa chuỗi ký tự sang số thập phân
- Khởi tạo số nguyên lớn
- Khởi tạo khóa công khai và khóa riêng tư hệ RSA
- Mã hóa và giải mã theo chuẩn RSA

2.1. Chuyển đổi, chuẩn hóa chuỗi ký tự sang số thập phân

Mục đích việc này là để chuyển đổi các chuỗi ký tự sang dãy số nguyên để mã hóa.

```
# Input 1: (String) decStr: Chuỗi ký tự OR danh sách thập phân
# Input 2: (int) flag: 0 = cờ chuyển sang danh sách thập phân, 1 = cờ chuyển sang chuỗi
# Output: (list) Danh sách thập phân OR (String) Chuỗi ký tự
def convertDecAndStr(decStr, flag):
    decStr_list = []
    if flag == 0:
      for i in decStr: decStr_list.append(ord(i)) # Duyệt từng ký tự trong chuỗi ký tự chuyển sang từng số nguyên
      return decStr_list
    else:
      for i in decStr: decStr_list.append(chr(i)) # Duyệt từng số nguyên trong danh sách số nguyên chuyển sang từng ký tự
      return ''.join(decStr_list)
```

2.2. Khởi tạo số nguyên lớn

Như đã đề cập ở trên, với sàng Eratosthenes và cách kiểm tra tính nguyên tố Rabin Miller. Ta sẽ khởi tạo số nguyên tố ngẫu nhiên trong giá trị n bit nhất định.

Ta cần phát sinh danh sách các số nguyên tố cho trước:

```
# Danh sách các số nguyên tố tạo trước để kiểm
first_primes_list = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
                     31, 37, 41, 43, 47, 53, 59, 61, 67,
                     71, 73, 79, 83, 89, 97, 101, 103,
                     107, 109, 113, 127, 131, 137, 139,
                     149, 151, 157, 163, 167, 173, 179,
                     181, 191, 193, 197, 199, 211, 223,
                     227, 229, 233, 239, 241, 251, 257,
                     263, 269, 271, 277, 281, 283, 293,
                     307, 311, 313, 317, 331, 337, 347, 349]
```

Kế đến, cần hàm để phát sinh 1 số lẻ ngẫu nhiên với *n* số bit cho trước

```
# Tạo 1 số lẻ có n bit trong khoảng 2**(n-1)+1 and 2**n-1
# Do tất cả các số nguyên tố (> 2) đều là số lẻ nên để có hiệu suất tốt hơn, chỉ có thể chọn số lẻ
# Input: (Int) n = số bit
# Output: (Int) 1 số lẻ ngẫu nhiên n bit
def nBitRandom(n):
    return random.randrange(2**(n-1)+1, 2**n - 1)
```

Tiếp theo ta cần hàm để kiểm tra tính nguyên tố ở mức độ thấp. Ta cần tính toán trước vài trăm số nguyên tố đầu tiên (như ở trên). Ta lấy số được phát sinh ngẫu nhiên (sau khi dùng hàm *nBitRandom*) chia lần lượt cho các số nguyên tố đầu tiên trong mảng *first_primes_list* đã tính toán để kiểm tra tính chia hết.

- Nếu chia hết, phép thử không thành công và phải tạo ra một số mới và lặp lại bước kiểm tra kiểm tra.
- Bước trên được lặp đi lặp lại miễn là tìm thấy một giá trị nguyên tố cùng nhau với tất cả các số nguyên tố trong danh sách số nguyên tố đã tạo

```
 # Input: (Int) n = số bit
 # Output: (Int) số nguyên tố cùng nhau (chỉ là kiểm bước 1)
def getLowLevelPrime(n):
    '''Generate a prime candidate divisible
    by first primes'''
    while True:
        # Lấy số ngẫu nhiên
        pc = nBitRandom(n)

        # Kiểm tra tính chia hết của các số nguyên tố đã cho trước
        for divisor in first_primes_list:
            if pc % divisor == 0 and divisor**2 <= pc:
                break
        # Nếu không tìm thấy số chia, trả về giá trị
        else: return pc
```

Sau khi vượt qua kiểm tra cấp thấp, tiếp theo ta sẽ kiểm tra lại tính nguyên tố của số tìm được một lần nữa bằng Rabin Miller. Đối với các số cực kỳ lớn, chẳng hạn như các số được sử dụng trong RSA, việc kiểm tra xác định xem giá trị được chọn có phải là số nguyên tố hay không là rất phi thực tế vì nó yêu cầu một lượng tài nguyên máy tính không hợp lý. Như vậy, cách tiếp cận xác suất số đó là số nguyên tố được ưu tiên hơn.

Khi qua một lần lặp lại của bài kiểm tra Rabin Miller, thì xác suất của số đó là số nguyên tố là 75%. Do đó, nếu một số vượt qua bài kiểm tra đủ số lần, ta có thể coi số đó là số nguyên tố với mức xác suất thoả mãn (ở đây ta thử với số lần lặp là 20 lần). Thông thường thì yêu cầu xác suất lỗi phải nhỏ hơn $(½)^{128}$

Hàm kiểm tra thỏa mãn bài kiểm tra Rabin Miller

```
 # Input:(Int) số nguyên tố cần kiểm tra
 # Output: (Bool) True = là số nguyên tố đúng, False = không là số nguyên tố
def isMillerRabinPassed(mrc):
    '''Run 20 iterations of Rabin Miller Primality test'''
    maxDivisionsByTwo = 0
    ec = mrc-1
    while ec % 2 == 0:
        ec >>= 1
        maxDivisionsByTwo += 1
    assert(2**maxDivisionsByTwo * ec == mrc-1)

    def trialComposite(round_tester):
        if pow(round_tester, ec, mrc) == 1:
            return False
        for i in range(maxDivisionsByTwo):
            if pow(round_tester, 2**i * ec, mrc) == mrc-1:
                return False
        return True

    # Số kiểm tra thử ở đây. Chạy 20 lần lặp lại bài kiểm tra số nguyên tố của Rabin Miller
    numberOfRabinTrials = 20
    for i in range(numberOfRabinTrials):
        round_tester = random.randrange(2, mrc)
        if trialComposite(round_tester):
            return False
    return True
```

Hàm tạo số nguyên tố lớn *n* bit thỏa mãn tất cả bài kiểm tra trên

```
def generatePrime(n=12):
    while True:
        prime_candidate = getLowLevelPrime(n)
        if not isMillerRabinPassed(prime_candidate):
            continue
        else: return prime_candidate
```

Nếu số ngẫu nhiên không thỏa mãn bài kiểm tra Rabin Miller thì ta tiếp tục tạo số mới và lặp lại bước kiểm tra tính nguyên tố đến khi thỏa mãn thì dừng và trả về số nguyên tố thỏa mãn yêu cầu.

2.3. Khởi tạo khóa công khai và khóa riêng tự chuẩn RSA

Để tạo ra cặp khóa công khai và khóa riêng tư, ta cần phát sinh ra n,e,d từ 2 số nguyên tố p và q (p và q phải khác nhau) được tạo ở trên

Khóa công khai tương ứng với cặp {e, n}

Trong đó, e phải: $1 < e < \phi(n)$ và nguyên tố cùng nhau với n và $\phi(n)$

$\phi(n) = (p-1)*(q-1)$ = số lượng nguyên tố cùng nhau với n

Khóa riêng tư tương ứng với cặp {d, n}

Trong đó, d phải thỏa mãn:

$de \ mod \ \phi(n) = 1$

Ta cần xây dựng các hàm sau để chuẩn bị cho công việc tạo khóa:

Hàm tìm ước chung lớn nhất giữa 2 số nguyên

```
# Hàm tìm ước chung lớn nhất 2 số a,b
# Input: (Int) a
# Input: (Int) b
# Output:(Int) ước chung lớn nhất
def gcd(a, b):
    while a:
        b, a = a, b % a
    return b
```

Hàm phát sinh n (Công thức: $n = p*q$)

```
# Hàm phát sinh n = p * q
# Input 1: (Int)số p nguyên tố bất kỳ
# Input 2: (Int) số q nguyên tố bất kỳ
# Output: (Int) n
def generateN(p, q):
    return p * q
```

Hàm phát sinh ϕ (Công thức: $\phi = (p-1) * (q-1)$)

```
# Hàm phát sinh phi
# Input 1: (Int) số p nguyên tố bất kỳ
# Input 2: (Int) số q nguyên tố bất kỳ
# Output: (Int) phi
def generatePhi(p, q):
    return (p-1) * (q-1)
```

Hàm phát sinh e (Chọn e ngẫu nhiên sao cho $1 < e < \phi(n)$)

```
# Khóa E
# Hàm phát sinh e (e với n  là khóa công khai)
# Input 1: (Int) số p nguyên tố bất kỳ
# Input 2: (Int) số q nguyên tố bất kỳ
# Output: (Int) e
def generateE(p, q):
    phi = generatePhi(p, q)

    for e in range(random.randrange(3, phi-1, 2), phi-1):
        if gcd(e, phi) == 1: return e
```

Hàm phát sinh d (Công thức: $de \bmod \phi(n) = 1$)

```
# Hàm phát sinh D (d với n này được coi là khóa riêng tư)
# Input : (Int) số e (Khóa E)
# Output: (Int) d
def generateD(e):
    phi = generatePhi(p, q)

    d = int(phi / e)
    while (True):
        if (d * e) % phi == 1: return d
        d += 1
```

Ví dụ kiểm thử:

```
# Tạo 2 số p, q nguyên tố ngẫu nhiên
p = generatePrime()
q = generatePrime()

# Nếu p mà trùng q, thì ta phải tạo lại q. Vì p và q phải khác nhau.
while (p == q):
    q = generatePrime()

print('Hai số nguyên tố ngẫu nhiên:')
print('\tSố nguyên tố thứ 1: ', p)
print('\tSố nguyên tố thứ 2: ', q)

n = generateN(p, q)
e = generateE(p, q)
d = generateD(e)

print('\nKhóa n: ', n)
print('khóa E: ', e)
print('Khóa D: ', d)

print('\nKhóa công khai {e, n}: {%d, %d}' %(e, n))
print('Khóa riêng tư {d, n}: {%d, %d}' %(d, n))
```

Ta có hai số nguyên tố ngẫu nhiên tìm được:

- Số nguyên tố thứ 1: 2503
- Số nguyên tố thứ 2: 3583

Sau khi chạy thuật toán trên tìm được:

- Khóa n: 8968249
- Khóa E: 1296689
- Khóa D: 2988821

Kết quả:

- Khóa công khai {e, n}: {1296689, 8968249}
- Khóa riêng tư {d, n}: {2988821, 8968249}

2.4. Thuật toán mã hóa và giải mã

Công thức để mã hóa và giải mã các tin nhắn là như nhau. Tất cả các tin nhắn được mã hóa bằng khóa công khai chỉ có thể được giải mã bằng khóa riêng tư và các tin nhắn được mã hóa bằng khóa riêng tư chỉ có thể được giải mã bằng khóa công khai.

Công thức mã hóa:

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

Công thức giải mã:

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

Hàm mã hóa và giải mã

```
def encryptAndDecrypt(m, key, n):
    res = 1
    m = m % n

    if (m == 0) : return 0

    while (key > 0) :
        # Nếu khóa là số lẻ, nhân khóa với kết quả
        if ((key & 1) == 1) : res = (res * m) % n

        # chìa khóa phải là chẵn
        key = key >> 1      # key = key/2
        m = (m * m) % n
    return res
```

Với:

- *m* tương ứng với số thập phân trong mảng thập phân đã được chuyển từ văn bản gốc
- *key* tương ứng **với** e và d trong khóa công khai hoặc khóa riêng tư
- *n* tương ứng với khóa *n*

CHƯƠNG 3

THỰC NGHIỆM

- Mã hóa bằng khóa công khai và giải mã bằng khóa riêng tư (Mã hóa tin)
- Mã hóa bằng khóa riêng tư và giải mã bằng khóa công khai (Chữ ký số)
- Mô tả thực nghiệm chương trình bằng video (đính kèm) và mã nguồn:
  https://github.com/ngdumitam277/PPNCKH_RSA

CHƯƠNG 4

KẾT LUẬN

4.1. Ưu điểm và khuyết điểm

- **Ưu điểm**

- RSA hỗ trợ trao đổi thông tin an toàn mà không cần khóa bí mật dùng chung.
- RSA là một thuật toán đã được thử nghiệm và chấp nhận sử dụng rộng rãi
- RSA sử dụng kích thước khóa lớn, thường là 2048 hoặc 4096 bit, khiến kẻ tấn công khó bẻ khóa bằng cách cố gắng dò tìm và thử mọi khó.
- RSA cũng có thể được sử dụng cho chữ ký điện tử, có thể giúp đảm bảo tính toàn vẹn và tính xác thực của dữ liệu.
- RSA có thể được sử dụng với các chế độ khác nhau như OAEP và PSS để cung cấp các tính năng bảo mật bổ sung.

- **Khuyết điểm**

- RSA chậm hơn so với các thuật toán mã hóa khóa đối xứng khi mã hóa hoặc giải mã lượng lớn dữ liệu.
- RSA dựa trên thực tế là rất khó tính toán để phân tích các số rất lớn. Tuy nhiên, những tiến bộ trong thuật toán phân tích như *Phương pháp sành trường số tổng quát – GNFS (General Number Field. Sieve)* đã giúp ta có thể phân tích các số lớn hơn dựa trên kích thước của khóa được sử dụng. Điều này có nghĩa là tính bảo mật của RSA phụ thuộc vào kích thước của khóa được sử dụng và kích thước khóa cần được tăng định kỳ để theo kịp những tiến bộ về sức mạnh tính toán.
- Việc sử dụng RSA để mã hóa dữ liệu công cộng, chẳng hạn như trong các hệ thống bỏ phiếu bầu cử, không được khuyến nghị do các lỗ hổng tiềm ẩn trong quá trình triển khai và quản lý hệ thống.
- RSA dễ bị Tấn công kênh bên (Side-Channel Attack), chẳng hạn như những cuộc tấn công khai thác rò rỉ thông tin từ việc triển khai phần cứng hoặc phần mềm của thuật toán.

4.2. Một số ứng dụng mã hóa RSA trong thực tế

- Truyền tải dữ liệu an toàn: RSA được sử dụng để truyền dữ liệu an toàn qua Internet, đặc biệt là trong các ứng dụng như ngân hàng trực tuyến, thương mại điện tử và email. Mục đích là tránh tình trạng nghe lén, theo dõi hoạt động cũng như lấy cắp dữ liệu cá nhân của người sử dụng
- Quản lý khóa: RSA được sử dụng để quản lý khóa trong các hệ thống bảo mật, chẳng hạn như chứng chỉ SSL và VPN. Khóa công khai được sử dụng để mã hóa và khóa riêng được sử dụng để giải mã.
- Quản lý giấy phép: RSA thường được sử dụng để tạo mã thông báo giấy phép để xác minh tính xác thực của phần mềm và ngăn chặn vi phạm bản quyền. Mã thông báo RSA có thể giúp xác định và xác thực người dùng mà không yêu cầu họ nhớ mật khẩu phức tạp.
- Chữ ký số / chữ ký điện tử: RSA được sử dụng để tạo chữ ký số xác minh tính xác thực của tin nhắn hoặc tài liệu. Ví dụ, trên các thẻ ATM luôn có phần chữ ký điện tử đã được mã hóa từ chữ ký của khách hàng khi đăng ký tài khoản tại ngân hang
- Ngoài ra còn RSA còn được ứng dụng vào trong công nghệ, kỹ thuật như: Công nghệ ARC trên cửa cuốn Austdoor,..

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Wikipedia (2022), *RSA (mã hóa)*

[2] Wikipedia (2022), *Sàng Eratosthenes*

[3] Wikipedia (2022), *Kiểm tra Miller-Rabin*

[4] Trần Thị Bình Minh (2021), *Tìm hiểu ứng dụng và giải thuật RSA*

Tiếng Anh

[3] GeeksforGeeks (2022), *How to generate Large Prime numbers for RSA Algorithm*

Địa chỉ trên Internet

1. https://vi.wikipedia.org/wiki/RSA_(m%C3%A3_h%C3%B3a)
2. https://vi.wikipedia.org/wiki/Ki%E1%BB%83m_tra_Miller-Rabin
3. https://vi.wikipedia.org/wiki/S%C3%A0ng_Eratosthenes
4. https://vnoi.info/wiki/translate/he/Number-Theory-2.md
5. https://www.geeksforgeeks.org/how-to-generate-large-prime-numbers-for-rsa-algorithm