TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**



**TIỂU LUẬN**

**MÔN HỌC: CẤU TRÚC RỜI RẠC**

**Mã môn học: 501044**

**TP. HỒ CHÍ MINH, THÁNG 4 NĂM 2023**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**



**TIỂU LUẬN**

**MÔN HỌC: CẤU TRÚC RỜI RẠC**

**Mã môn học: 501044**

Họ và tên sinh viên: TRẦN DUY KHỞI

Mã số sinh viên: 52100809

Ngành học: Kỹ thuật phần mềm

Email: 52100809@student.tdtu.edu.vn

**TP. HỒ CHÍ MINH, THÁNG 4 NĂM 2023**

**LỜI CẢM ƠN**

Môn học Cấu Trúc Rời Rạc học kỳ này trải qua dưới hình thức 50% online – 50% offline, đó là một sự khó khăn rất lớn đối với cả sinh viên và giảng viên trong vấn đề tiếp thu và truyền đạt tri thức. Vì thế để hoàn thành được bài tiểu luận môn Cấu Trúc Rời Rạc, em đã được giảng viên Nguyễn Quốc Bình tận tình hướng dẫn. Em muốn gửi lời cảm ơn đến thầy vì đã nhiệt tình giúp đỡ em hoàn thành bài tiểu luận này.

Em kính chúc thầy và gia đình luôn luôn có thật nhiều sức khỏe đặc biệt là trong thời gian dịch bệnh khó khăn này, để tiếp tục sự nghiệp dẫn dắt và truyền đạt kiến thức tới các thế hệ sinh viên như chúng em.

Mục Lục

[Chương 1: Tìm modulo nghịch đảo n. 5](#_Toc132275770)

[1. Tìm hiểu về Modulo nghịch đảo n 5](#_Toc132275771)

[a. Định nghĩa. 5](#_Toc132275772)

[b. Ứng dụng của Modulo nghịch đảo n. 5](#_Toc132275773)

[2. Tìm hiểu về thuật toán Euclide mở rộng và cách tính modulo nghịch đảo n bằng thuật toán Euclide mở rộng. 6](#_Toc132275774)

[a. Thuật toán Euclide mở rộng. 6](#_Toc132275775)

[b. Tính Modulo nghịch đảo n bằng thuật toán Euclide mở rộng. 6](#_Toc132275776)

[3. Xây dựng chương trình Python tìm Modulo nghịch đảo n sử dụng thuật toán Euclide mở rộng. 8](#_Toc132275777)

[a. Phần code python và giải thích code. 8](#_Toc132275778)

[b. Kiểm tra tính đúng đắn của code và xác minh kết quả. 11](#_Toc132275779)

[Chương 2: Hệ thống mật mã RSA. 16](#_Toc132275780)

[1. Tìm hiểu về hệ thống mật mã RSA. 16](#_Toc132275781)

[2. Triển khai bằng code Python. 20](#_Toc132275782)

[a. Triển khai code. 20](#_Toc132275783)

[b. Giải thích code. 21](#_Toc132275784)

[3. Kiểm tra tính đúng đắn và xác minh kết quả của code. 23](#_Toc132275785)

[4. Tìm hiểu về hiệu quả mà hệ thống mật mã RSA đem lại. 25](#_Toc132275786)

[5. Tìm hiểu về những điểm hạn chế của hệ thống mật mã RSA. 26](#_Toc132275787)

[6. Cách để cải thiện hệ thống mật mã RSA. 27](#_Toc132275788)

[TÀI LIỆU THAM KHẢO 29](#_Toc132275789)

[PHIẾU TỰ ĐÁNH GIÁ 30](#_Toc132275790)

Chương 1: Tìm modulo nghịch đảo n.

1. Tìm hiểu về Modulo nghịch đảo n
2. Định nghĩa.

- Ta xét một số nguyên dương m và các số nguyên trên modulo m (các số từ 0 đến m – 1)

- Với một số nguyên *a* thì ta sẽ gọi modulo nghịch đảo m (modular multiplicative inverse) của a (ký hiệu: ) là một số nguyên thỏa mãn:

\* Lưu ý:

+ tồn tại khi và chỉ khi a và modulo m là hai số nguyên tố cùng nhau (không có ước chung nào khác ngoài 1). Nếu như a và modulo m không phải là hai số nguyên tố cùng nhau thì sẽ không tồn tại. Ví dụ, với m = 6 và a = 2 thì ta sẽ không thể tìm được thỏa mãn

+ Nếu như gcd(a, m) = 1 thì ta sẽ luôn luôn tìm được

1. Ứng dụng của Modulo nghịch đảo n.

- Modulo nghịch đảo có nhiều ứng dụng trong nhiều lĩnh vực khác nhau, trong đó một số ứng dụng mà ta có thể kể đến như:

+ Mã hóa và giải mã thông tin.

+ Kiểm tra một số có phải số nguyên tố hay không.

+ Giải các phương trình tuyến tính và tính toán trong đại số tuyến tính.

+ Modulo nghịch đảo cũng được sử dụng trong việc tính toán đồ thị ví dụ tìm đường đi ngắn nhất giữa hai đỉnh.

+ …

1. Tìm hiểu về thuật toán Euclide mở rộng và cách tính modulo nghịch đảo n bằng thuật toán Euclide mở rộng.
2. Thuật toán Euclide mở rộng.

- Thuật toán Euclide mở rộng được dùng để giải một phương trình vô định nguyên hay còn gọi là phương trình Diophantine có dạng như sau:

\* Trong đó thì a, b, c là các số nguyên đã cho và x, y là biến mà ta cần phải đi tìm.

- Theo đinh lý Bézout: Cho hai số nguyên a và b thì khi đó sẽ luôn tồn tại hai số x và y sao cho:

- Người ta cũng chứng minh được rằng phương trình trên có nghiệm khi và chỉ khi gcd (a, b) = c. Như vậy cũng có nghĩa là phương trình Diophantine có thể có vô số nghiệm, và nếu như có vô số nghiệm thì từ mỗi một nghiệm ta có thể sinh ra được những nghiệm khác.

b. Tính Modulo nghịch đảo n bằng thuật toán Euclide mở rộng.

- Một trong những ứng dụng quan trọng nhất của thuật toán Euclide mở rộng đó chính là dùng để tính modulo nghịch đảo. Với a và m là hai số nguyên tố cùng nhau (gcd (a, m) = 1) thì luôn tồn tại nghịch đảo của a modulo m:

- Từ đây thì ta có thể thấy được rằng tồn tại b sao cho:

- Đây cũng chính là kết quả của thuật toán Euclide mở rộng: cho a và b, thuật toán sẽ tìm các giá trị x, y sao cho:

- Với c = gcd (a, b). Ngoài việc tìm c thì ta cần phải lưu lại các trọng số liên quan. Nếu như c = gcd (a, b) = 1 thì a và b là hai số nguyên tố cùng nhau, lúc này công thức này giống với công thức ở bên trên, có nghĩa là x sẽ là nghịch đảo của a modulo b và tương tự thì y là sẽ nghịch đảo của b modulo a.

- Tuy nhiên, có một lưu ý nhỏ khi tính nghịch đảo modulo đó là giá trị nghịch đảo modulo luôn là một số không âm trong phạm vi từ 0 đến b – 1 với b là số modun cho trước. Ngoài ra nếu như một trong hai số a và b bằng 1 hoặc một trong hai số a và b bằng 0 thì cũng không tồn tại nghịch đảo modulo.

- Để dễ hiểu hơn thì ta sẽ đi vào phần ví dụ:

+ Thuật toán Euclide áp dụng cho 340 và 17 sẽ là:

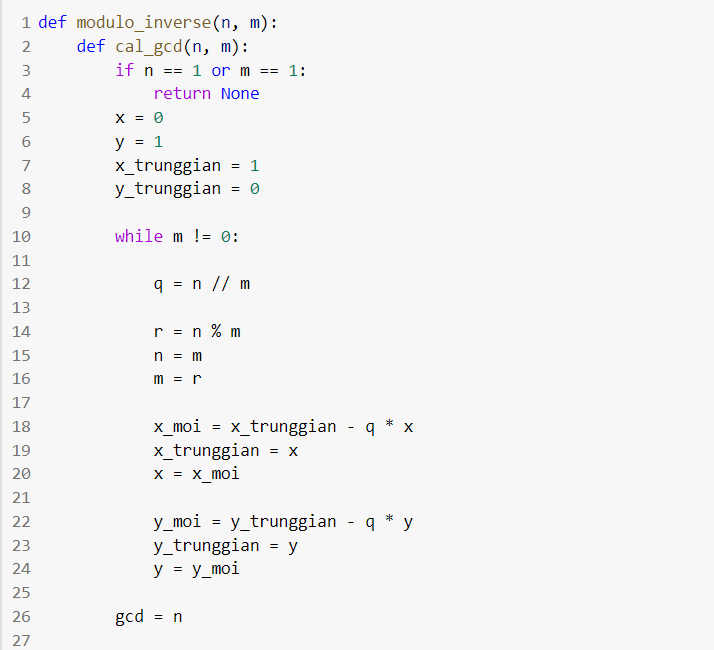
+ Ta có:

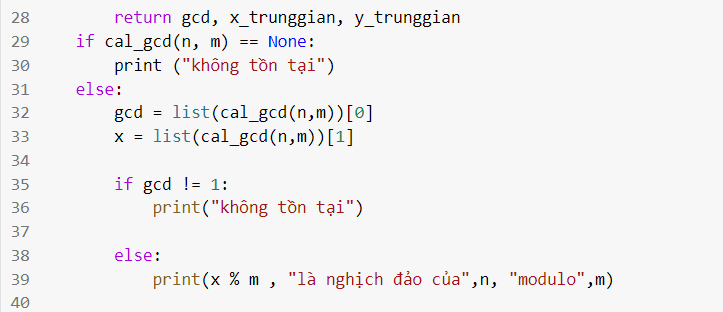
+ Mà modulo là giá trị dương nên ta sẽ lấy -5 % 27 = 22. Đây là nghịch đảo của 340 modulo 27

+ Vậy 22 là nghịch đảo của 340 modulo 27

1. Xây dựng chương trình Python tìm Modulo nghịch đảo *n* sử dụng thuật toán Euclide mở rộng.
2. Phần code python và giải thích code.

\* Phần code python:





\* Giải thích từng dòng mã code:

+ Dòng 1: Khai báo hàm “modulo\_inverse” với hai tham số đầu vào là n và m. Đây chính là 2 số nguyên mà ta đang cần tính modulo nghịch đảo

+ Dòng 2 – 28: Định nghĩa hàm con “cal\_gcd” với hai tham số đầu vào là n và m để tính gcd của hai số nguyên n và m. Các biến x, y, x\_trunggian, y\_trunggian để lưu trữ các giá trị trung gian trong quá trình tính gcd bằng thuật toán Euclide mở rộng.

+ Dòng 3 – 4: Câu lệnh if để kiểm tra xem nếu như một trong hai tham số đầu vào mà bằng 1 thì hàm “cal\_gcd” sẽ trả về kết quả là None, tức là kiểm tra xem nghịch đảo modulo có tồn tại hay không.

+ Dòng 10 – 24: Sử dụng vòng lặp while để tính gcd bằng thuật toán Euclide mở rộng. Biến q dùng để lưu phần nguyên của phép chia n cho m, biến r dùng để lưu phần dư của phép chia n cho m và các biến x, x\_trunggian, y, y\_trunggian dùng để cập nhật giá trị tính toán sau mỗi vòng lặp

+ Dòng 26: Dùng để gán giá trị của n là gcd hai số n và m sau khi đã hoàn thành vòng lặp while  
+ Dòng 28: Trả về các giá trị gcd, x\_trunggian, y\_trunggian dưới dạng một tuple

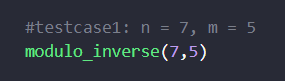
+ Dòng 29 – 30: Kiểm tra xem hàm “cal\_gcd” có trả về None hay không, nếu có thì in ra thông báo “không tồn tại”. Ngược lại, nếu làm “cal\_gcd” không trả về None thì sẽ thực hiện các câu lệnh từ dòng 32 tới dòng 39.  
+ Dòng 32 – 33: Gọi hàm “cal\_gcd” và lưu trữ giá trị gcd vào biến gcd, giá trị x\_trunggian vào biến x

+ Dòng 35 – 36: Nếu như gcd không bằng 1 thì trả về kết quả không tồn tại

+ Dòng 38 – 39: Nếu như gcd bằng 1 thì trả về kết quả của phép chia lấy phần dư của x cho m, đây chính là nghịch đảo của n modulo m.

1. Kiểm tra tính đúng đắn của code và xác minh kết quả.

+ testcase 1: n = 7, m = 5. Kết quả mong đợi là: “3 là nghịch đảo của 7 modulo 5”.





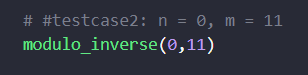
Giải thích:

Ta viết:

là nghịch đảo của 7 modulo 5 vì nghịch đảo modulo là số dương

vậy 3 là nghịch đảo của 7 modulo 5

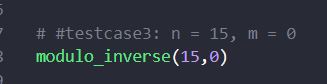
+ testcase 2: n = 0, m = 11. Kết quả mong đợi “Không tồn tại”





Giải thích: vì không có số nào nhân với 0 cho ra kết quả khác 0

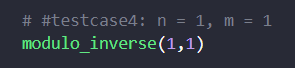
+ testcase 3: n = 15, m = 0. Kết quả mong đợi “Không tồn tại”

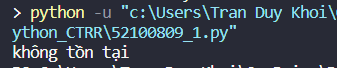




Giải thích: vì không có số nào nhân với 0 cho ra kết quả khác 0

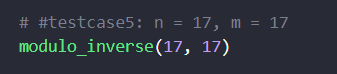
+ testcase 4: n = 1, m = 1. Kết quả mong đợi: “Không tồn tại”

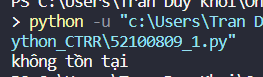




Giải thích: 1 không phải là số nguyên tố

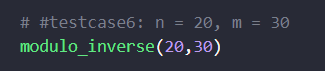
+ testcase 5: n = 17, m = 17. Kết quả mong đợi: “Không tồn tại”





Giải thích: không tồn tại modulo nghịch đảo của 2 số giống nhau

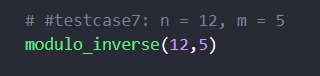
+ testcase 6: n = 20, m = 30. Kết quả mong đợi “Không tồn tại”

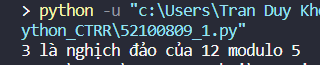




Giải thích: vì đây là 2 số chẵn, không phải là hai số nguyên tố cùng nhau

+ testcase 7: n = 12, m = 5. Kết quả mong đợi “3 là nghịch đảo của 12 modulo 5”



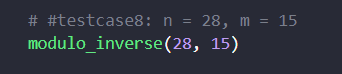


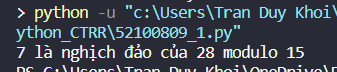
Giải thích:

Ta viết:

là nghịch đảo của 12 modulo 5 vì modulo nghịch đảo phải là số nguyên dương.

+ testcase 8: n = 28, m = 15. Kết quả mong đợi “7 là nghich đảo của 28 modulo 15”



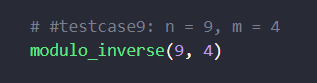


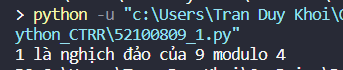
Giải thích:

Ta viết:

Vậy 7 là nghịch đảo của 28 modulo 15

+ testcase 9: n = 9, m = 4. Kết quả mong đợi “1 là nghịch đảo của 9 modulo 4”



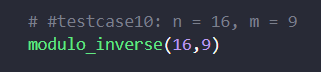


Giải thích:

Ta viết:

Vậy 1 là nghịch đảo 9 modulo 4

+ testcase 10: n = 16, m = 9. Kết quả mong đợi: “4 là nghịch đảo của 16 modulo 9”





Giải thích:

Ta viết:

Vậy 4 là nghịch đảo của 16 modulo 9.

Chương 2: Hệ thống mật mã RSA.

1. Tìm hiểu về hệ thống mật mã RSA.

- RSA là một hệ mã hóa bất đối xứng được phát triển bởi Ron Rivest, Adi Shamir và Leonard Adleman (cái tên RSA cũng chính là tên viết tắt của 3 tác giả này) và nó được sử dụng rộng rãi trong công tác mã hoá và công nghệ chữ ký điện tử. Trong hệ mã hóa này, khóa công khai có thể chia sẻ công khai cho tất cả mọi người. Hoạt động của RSA dựa trên 2 giai đoạn chính:

\* Giai đoạn 1: Tính cặp khóa công khai và khóa bí mật. Giai đoạn này gồm 6 bước:

+ Bước 1: Chọn hai số nguyên tố p và q.

+ Bước 2: Ta sẽ tính n bằng tích của p và q. Trong đo thì n được gọi là modun mã hóa và giải mã

+ Bước 3: Ta sẽ tính

+ Bước 4: Chọn một số nguyên e làm khóa công khai sao cho e thỏa mãn điều kiện sau: 1 < e < và e là số nguyên tố cùng nhau với

+ Bước 5: Tìm khóa bí mật d bằng cách sử dụng thuật toán Euclide mở rộng sao cho (d.e) mod = 1

+ Bước 6: Xác định cặp khóa công khai và khóa bí mật.

Cặp khóa công khai là (e, n) và cặp khóa bí mật là (d, n)

Sau khi đã tìm được cặp khóa công khai và khóa bí mật thì người gửi sẽ sử dụng khóa công khai để mã hóa thông điệp sau đó người nhận sẽ sử dụng khóa bí mật để giải mã thông điệp đã được mã hóa

\* Giai đoạn 2: Tiến hành mã hóa và giải mã thông điệp sử dụng cặp khóa công khai và khóa bí mật. Giai đoạn này gồm 3 bước:

+ Bước 1: Chuẩn bị các khóa và thông điệp cần mã hóa

+ Bước 2: Mã hóa thông điệp. Ở bước này ta sẽ chuyển thông điệp cần mã hóa sang dạng số, thông thường thì sẽ sử dụng bảng mã ASCII để biểu diễn thông điệp. Ví dụ, chuyển thông điệp “hello” thành các số trong bảng mã ASCII sẽ là: “h” – 104, “e” – 101, “l” – 108, “l” – 108, “o” – 111. Sau khi chuyển thì ta sẽ mã hóa thông điệp sử dụng công thức sau:

Trong đó: C là mã hóa của thông điệp

M là số đại diện cho thông điệp

e là số nguyên được chọn làm khóa công khai ở bước 4 – GD1

n là modun mã hóa và giải mã ở bước 1 – GD1

+ Bước 3: Tiến hành giải mã thông điệp. Ở bước này ta sẽ sử dụng công thức:

Trong đó: C là mã hóa của thông điệp

M là số đại diện cho thông điệp

d là số nguyên được chọn làm khóa bí mật ở bước 5 – GD1

n là modun mã hóa và giải mã ở bước 1 – GD1

- Để hiểu rõ hơn thì ta sẽ đi vào phần ví dụ:

Tìm cặp khóa công khai và khóa bí mật bằng thuật toán RSA sau đó mã hóa và giải mã thông điệp “hello”.

Giai đoạn 1:

+ Bước 1: Chọn hai số nguyên tố lớn là p và q (vì là ví dụ nên ta sẽ chọn p = 11 và q = 41 để cho dễ tính)

p = 11

q = 41

+ Bước 2: Ta dễ dàng tính được modun mã hóa và giải mã

n = p.q = 11.41 = 451

+ Bước 3: Tính = (p – 1) (q – 1) = 10.40 = 400

+ Bước 4: Chọn 1 số nguyên e làm khóa công khai sao cho e thỏa mãn 1 < e < và e là số nguyên tố cùng nhau với . Ta chọn e = 3 vì gcd(3, 400) = 1 và 1 < 3 <= 400

+ Bước 5: Tìm khóa bí mật d sử dụng thuật toán Euclide mở rộng sao cho (d.e) mod = 1 (d.3) mod 400 = 1

Ta dễ dàng tính được d = 267

+ Bước 6: Xác định cặp khóa công khai và khóa bí mật

Cặp khóa công khai là (e, n) = (3, 451)

Cặp khóa bí mật là (d, n) = (267, 451)

Giai đoạn 2:

+ Bước 1: chuẩn bị các khóa và thông điệp:

Cặp khóa công khai: (e, n) = (3, 451)

Cặp khóa bí mật: (d, n) = (267, 451)

Thông điệp cần mã hóa: “hello”

+ Bước 2: Tiến hành mã hóa.

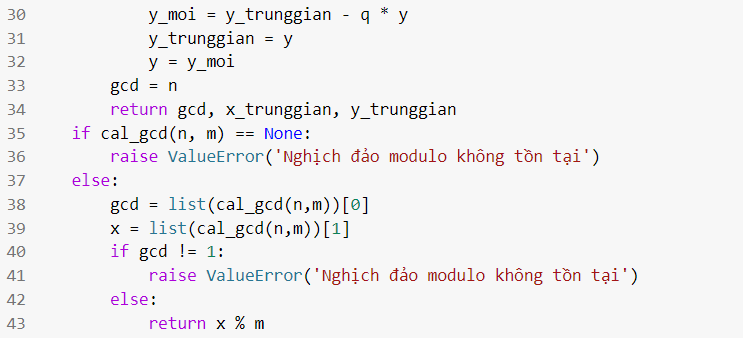
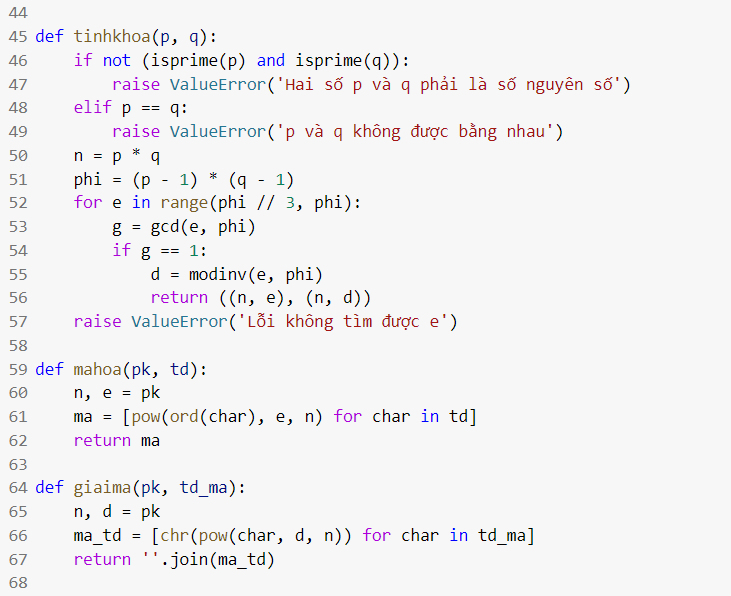
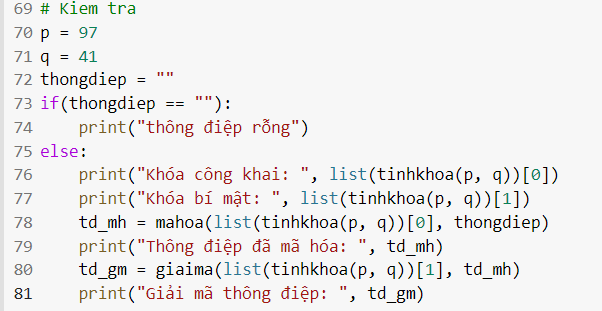
Đầu tiên ta sẽ chuyển đổi thông điệp “hello” sang số theo bảng mã ASCII:

“hello” sẽ thành [104, 101, 108, 108, 111]. Sau đó ta sử dụng công thức:

+ Bước 3: Tiến hành giải mã thông điệp. Ta sử dụng công thức:

Sau khi giải mã thông điệp thì ta sẽ được dãy [104, 101, 108, 108, 111] và ta tiến hành đổi số trong bảng mã ASCII về chữ sẽ được “hello”

1. Triển khai bằng code Python.
2. Triển khai code.

1. Giải thích code.

- Dòng 2: import hàm isprime từ thư viện sympy. Hàm này dùng để kiểm tra xem một số có phải số nguyên tố hay không.

- Dòng 4 – 7: Khai báo hàm gcd nhận vào 2 tham số đầu vào là a và b.

- Dòng 5 – 7: Vòng lặp while được sử dụng để tính gcd của 2 số a và b bằng thuật toán Euclide.

- Dòng 9: Khai báo hàm “modulo\_inverse” với hai tham số đầu vào là n và m. Đây chính là 2 số nguyên mà ta đang cần tính modulo nghịch đảo

- Dòng 10 – 33: Định nghĩa hàm con “cal\_gcd” với hai tham số đầu vào là n và m để tính gcd của hai số nguyên n và m. Các biến x, y, x\_trunggian, y\_trunggian để lưu trữ các giá trị trung gian trong quá trình tính gcd bằng thuật toán Euclide mở rộng.

- Dòng 11 – 12: Câu lệnh if để kiểm tra xem nếu như một trong hai tham số đầu vào mà bằng 1 thì hàm “cal\_gcd” sẽ trả về kết quả là None, tức là kiểm tra xem nghịch đảo modulo có tồn tại hay không.

- Dòng 18 – 32: Sử dụng vòng lặp while để tính gcd bằng thuật toán Euclide mở rộng. Biến q dùng để lưu phần nguyên của phép chia n cho m, biến r dùng để lưu phần dư của phép chia n cho m và các biến x, x\_trunggian, y, y\_trunggian dùng để cập nhật giá trị tính toán sau mỗi vòng lặp

- Dòng 33: Dùng để gán giá trị của n là gcd hai số n và m sau khi đã hoàn thành vòng lặp while  
- Dòng 34: Trả về các giá trị gcd, x\_trunggian, y\_trunggian dưới dạng một tuple

- Dòng 35 – 36: Kiểm tra xem hàm “cal\_gcd” có trả về None hay không, nếu có thì in ra thông báo “không tồn tại”. Ngược lại, nếu làm “cal\_gcd” không trả về None thì sẽ thực hiện các câu lệnh từ dòng 37 tới dòng 43.  
- Dòng 38 – 39: Gọi hàm “cal\_gcd” và lưu trữ giá trị gcd vào biến gcd, giá trị x\_trunggian vào biến x

- Dòng 40 – 41: Nếu như gcd không bằng 1 thì trả về kết quả không tồn tại

- Dòng 42 – 43: Nếu như gcd bằng 1 thì trả về kết quả của phép chia lấy phần dư của x cho m, đây chính là nghịch đảo của n modulo m.

- Dòng 45 – 57: khai báo hàm “tinhkhoa” nhận 2 tham số đầu vào là p và q.

- Dòng 46 – 47: kiểm tra nếu p và q không phải là số nguyên tố thì raise ra lỗi “Hai số p và q phải là số nguyên tố”. Bởi vì đây là thuật toán RSA nên bắt buộc p và q phải là hai số nguyên tố.

- Dòng 48 – 49: kiểm tra xem p và q có bằng nhau hay không, nếu bằng nhau thì raise lỗi “p và q không được bằng nhau”

- Dòng 50: tính modun mã hóa và giải mã n bằng p nhân q.

- Dòng 51: tính = (p – 1)(q – 1)

- Dòng 52 – 56: vòng lặp for chạy từ chia 3 tới để tìm e làm khóa công khai

- Dòng 57: nếu không tìm được e thì raise lỗi “Lỗi không tìm được e”

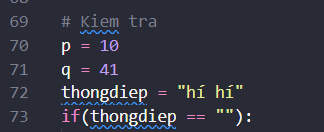
- Dòng 59 – 62: khai báo hàm “mahoa”. Hàm này sẽ nhận 2 giá trị đầu vào là pk ở dạng tulple và td là thông điệp cần mã hóa ở dạng chuỗi, pk chính là cặp khóa công khai (e, n). Sau đó hàm này sẽ tiến hành mã hóa thông điệp bằng phương pháp mã hóa RSA.

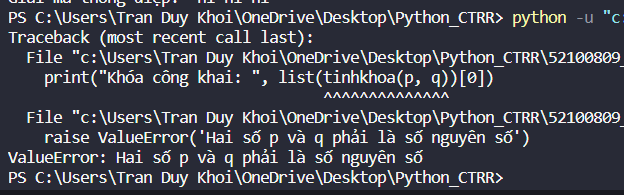
- Dòng 64 – 67: khai báo hàm “giaima”. Hàm này nhận 2 giá trị đầu vào đó là pk ở dạng tuple và td\_ma là thông điệp đã mã hóa và đang cần được giải mã, pk chính là cặp khóa bí mật (d, n). Sau đó hàm này dùng để giải mã thông điệp bằng phương pháp mã hóa RSA.

- Dòng 70 – 81: dùng để kiểm tra kết quả đoạn code.

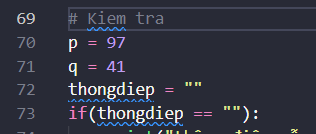
1. Kiểm tra tính đúng đắn và xác minh kết quả của code.

+ testcase 1: p hoặc q không phải là số nguyên tố. Kết quả mong đợi “Hai số p và q phải là hai số nguyên tố”



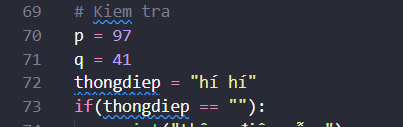


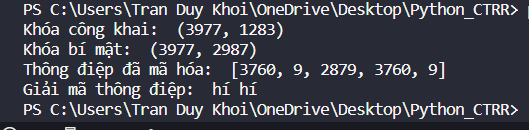
+ testcase 2: thông điệp rỗng. Kết quả mong đợi “thông điệp rỗng”





+ testcase 3: p và q đúng, thông điệp không rỗng. Kết quả mong đợi là sẽ trả về được khóa bí mật, khóa công khai, thông điệp đã được mã hóa và thông điệp sau khi được giải mã.





1. Tìm hiểu về hiệu quả mà hệ thống mật mã RSA đem lại.

Hệ thống mật mã RSA có nhiều hiệu quả, bao gồm:

- Thứ nhất phải kể đến đó là sự bảo mật cao. RSA là một trong những thuật toán mã hóa công khai được sử dụng rộng rãi vì tính bảo mật mà nó đem lại. Nó dựa trên một bài toán phân tích nguyên tố, đây được coi là một bài toán rất khó để giải quyết cho những số lớn, đồng nghĩa với việc khó có thể giải mã được dữ liệu nếu như không có khóa bí mật.

- Thứ hai đó là RSA là một hệ thống mật mã có tính đối xứng giữa khóa công khai và khóa bí mật. Khóa công khai dùng để mã hóa dữ liệu còn khóa bí mật dùng để giải mã dữ liệu. Điều này sẽ cho phép bất cứ ai cũng có thể sử dụng khóa công khai đã mã hóa và gửi tin nhắn cho người nhận. Những một điều hiển nhiên là người sở hữu khóa bí mật mới giải mã được mà thôi. Thường thì cặp khóa này được sinh ra đảm rằng rằng từ khóa công khai sẽ rất khó (gần như là không thể) tìm ra được khóa bí mật. Vì vậy, bất cứ kẻ tấn công nào nếu có được khóa công khai cũng không thể có được khóa bí mật để giải mã.

- Thứ ba đó là tin nhắn được gửi từ chủ của khóa bí mật tạo bằng RSA rất khó để có thể giả mạo vì nếu thay đổi tin nhắn thì giá trị hash cũng phải thay đổi theo. Những kẻ nghe lén trong mạng tất nhiên sẽ có thể đọc tin nhắn gốc và hash của nó nhưng hắn sẽ không thể nào thay đổi được tin nhắn vì không có khóa bí mật để sửa đổi chữ ký số cho phù hợp.

- Thứ tư là RSA đã được kiểm chứng và được sử dụng trong thực tế nhiều năm vì nó đáng tin cậy. Nó được sử dụng rộng rãi trong nhiều ứng dụng, chẳng hạn như: trong các giao thức mạng, trong kỹ thuật số, ngân hàng trực tuyến, chữ ký điện tử, …

- Thứ năm đó là chữ ký số. RSA cho phép người dùng xác thực tính toàn vẹn của dữ liệu đồng thời đảm bảo được tính xác thực của người gửi.

- Cuối cùng đó là tính linh hoạt và nhất quán. RSA có tính linh hoạt cao trong việc thay đổi kích thước của khóa, đồng thời nó cũng đảm bảo tính nhất quán trong quá trình mã hóa trên nhiều nền tảng và nhiều ứng dụng khác nhau.

1. Tìm hiểu về những điểm hạn chế của hệ thống mật mã RSA.

Hệ thống mật mã RSA hiện nay đang là một trong những phương pháp mã hóa phổ biến và được sử dụng rộng rãi trong các ứng dụng bảo mật thông tin. Tuy nhiên, nó cũng đang phải đối mặt với một số những hạn chế như sau:

- Thứ nhất phải kể đến đó là về tốc độ xử lý của hệ thống mật mã RSA: Công việc mã hóa và giải mã trong hệ thống mật mã RSA đòi hỏi phải thực hiện các phép tính có độ phức tạp cao, đặc biệt là trong quá trình mã hóa và giải mã dữ liệu lớn. Do đó, tốc độ xử lý của thuật toán này thường chậm hơn so với những thuật toán mã hóa bất đối xứng khác.

- Thứ hai đó là các thuật toán sinh khóa RSA hiện này chỉ phòng được các kỹ thuật crack thông thường sử dụng bằng máy tính cá nhân. Tuy nhiên, với sự phát triển của các siêu máy tính cũng như các máy tính lượng tử hiện nay thì hệ mã hóa bất đối xứng mà RSA đang sử dụng phải đối mặt với rất nhiều thách thức vì nó có thể bị phá bởi các thuật toán lượng tử.

- Thứ ba đó là hệ thống mật mã RSA yêu cầu việc lưu trữ và quản lý khóa bí mật một cách an toàn để đảm bảo tính bảo mật của những dữ liệu đã được mã hóa. Nếu như khóa bí mật bị rò rỉ hoặc bị đánh cắp thì hệ thống mật mã RSA sẽ bị đe dọa.

- Thứ tư, mặc dù RSA là một thuật toán mã hóa mạnh nhưng nếu như độ dài của khóa trong hệ thống mật mã RSA không đủ lớn thì nó rất dễ bị phá bằng cách sử dụng các kỹ thuật tấn công như brute-force, phân tích nguyên tố, ...

- Thứ năm, kích thước của khóa trong hệ thống mật mã RSA có thể rất lớn, đặc biệt là trong những ứng dụng đòi hỏi tính bảo mật cao. Tuy nhiên thì nếu như một khóa có độ dài lớn thì đồng nghĩa với việc phải tăng thời gian tính toán để mã hóa và giải mã. Chính vì vậy nếu như thuật toán này sử dụng trong các ứng dụng có yêu cầu về tốc độ cao như là giao tiếp mạng thì nó sẽ làm giảm đi đáng kể hiệu suất của hệ thống.

- Cuối cùng, một trong những bước quan trọng nhất của hệ thống mật mã RSA đó là việc tìm và chọn các số nguyên tố lớn để sử dụng làm khóa công khai và khóa bí mật. Do đó nếu như có một thuật toán hiệu quả cho việc phân tích số nguyên tố thì hệ thống mật mã RSA sẽ bị đánh bại. Hiện nay thì vẫn chưa có thể giải quyết được bài toán này trên những máy tính hiện đại nhưng với sự phát triển mạnh mẽ của công nghệ thông tin thì việc tìm ra một thuật toán để giải quyết bài toán phân tích số nguyên tố trong tương lai là một điều hoàn toàn có thể xảy ra.

1. Cách để cải thiện hệ thống mật mã RSA.

Một số cách để cải thiện hệ thống mật mã RSA bao gồm:

- Thứ nhất, kích thước của khóa ở trong hệ thống mật mã RSA được coi là một hạn chế vì nó đòi hỏi tốc độc tính toán và khả năng xử lý của hệ thống. Vì vậy để đạt được sự cân bằng về tính bảo mật và hiệu quả mà hệ thống mật mã RSA đem lại thì người dùng cần lựa chọn kích thước của khóa sao cho phù hợp với ứng dụng và phù hợp với môi trường triển khai hệ thống mật mã RSA để đạt được sự cân bằng giữa tính bảo mật và hiệu quả.

- Thứ hai, sự phụ thuộc vào tính nguyên tố của các số là hạn chế của hệ thống mật mã RSA. Để giảm bớt rủi rỏ này thì người dùng cần phải chọn các số nguyên tố lớn có đủ độ dài để đảm bảo tính bảo mật của hệ thống và đồng thời cũng phải luôn luôn cập nhật những công nghệ mã hóa và phân tích nguyên tố mới nhất để có thể tránh được những cuộc tấn công về bảo mật trong tương lại.

- Cuối cùng, chúng ta phải sử dụng phương pháp mã hóa kết hợp. Ngoài sử dụng hệ thống mật mã RSA thì người dùng cần kết hợp sử dụng các phương pháp mã hóa khác. Ví dụ như mã hóa đối xứng hoặc mã hóa đối xứng kết hợp với mã hóa công khai để đạt được tính bảo mật cao hơn và hạn chế được thấp nhất những cuộc tấn công tiềm ẩn.

TÀI LIỆU THAM KHẢO

[1]. Susanna S. Epp, [2011], Discrete Mathematics with applications, 4th edition, Brooks/Cole, Boston.

[2]. Kenneth H. Rosen, [2012], Discrete mathematics and its applications, 7th edition, McGraw-Hill, New York.

[3]. Allan Stavely, [2014], Programming and mathematical thinking: A gentle introduction to discrete math featuring Python, The New Mexico Tech Press, Socorro, New Mexico.

[4]. R. P. Grimaldi, [2004], Discrete and Combinatorial Mathematics: An Applied Introduction, 5th edition, Pearson Education, Boston.

[5]. Rowan Garnier, John Taylor, [2002], Discrete Mathematics for New Technology, 2nd edition, Institute of Physics, Bristol, Philadelphia.

PHIẾU TỰ ĐÁNH GIÁ

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Criterial | | Scale | 0 score | ½ score | Full score | Sefl-evaluation | Reason |
| Task 1 | Theorical research | 1 | Do nothing or wrongly | Not enough details, no example, no comment | Correct calculations, detailed explanations | 1 | Correct calculations, detailed explanations |
| Implementation | 2 | Error | Correct but bad performance | Correct and good performance | 2 | Correct and good performance |
| Test | 1 | No test | Test without verication | Test and verification | 1 | Test and verification |
| Task 2 | Theorical research | 2 | Do nothing or wrongly | Not enough details, no example, no comment | Correct calculations, detailed explanations | 2 | Correct calculations, detailed explanations |
| Implementation | 1 | Error | Correct but bad performance | Correct and good performance | 1 | Correct and good performance |
| Test | 1 | No test | Test without verication | Test and verification | 1 | Test and verification |
| Analysis | 0.5 | Do nothing or wrongly | Not enough details, no example, no comment | Correct, detailed explanations | 0.5 | Correct, detailed explanations |
| Discussion | 0.5 | Do nothing or wrongly | Not enough details, no example, no comment | Correct, detailed explanations | 0.5 | Correct, detailed explanations |
| Recommendation | 0.5 | Do nothing or wrongly | Not enough details, no example, no comment | Correct, detailed explanations | 0.5 | Correct, detailed explanations |
| Rerference | | 0.5 | No reference | Wrong format | Right format | 0.5 | Right format |
| Total | | 10 | Result | | | 10 |  |