BỘ GIÁO DỤC – ĐÀO TẠO

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN ĐHQGHCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

Môn: Nhập môn mã hoá – mật mã

Bài tập nhóm

19127449 – Phùng Anh Khoa

19127525 – Nguyễn Thanh Quân

TP.HCM, ngày 11 tháng 12 năm 2021

Mục lục

I. Lý thuyết

1. Giới thiệu

Để hiểu rõ hơn về thuật toán kiểm tra số nguyên tố ta cần hiểu số nguyên tố là gì và tại sao lại có thuật toán kiểm tra số nguyên tố.

Số nguyên tố là số tự nhiên lớn hơn 1 và chỉ có hai ước số là 1 và chính nó, như vậy các số 2, 3, 5, 7, 11, … được coi là các số nguyên tố. Kiểm tra một số có phải là một số nguyên tố hay không là một bài toán đơn giản mà bất kỳ một lập trình viên nào cũng đã làm. Tuy nhiên, Để đáp ứng nhu cầu về bảo mật của ngành mật mã hoá, những thuật toán kiểm tra số nguyên tố phải xử lý được cái số có số bit rất lớn mà các thuật toán đơn giản không thể xử lý được hoặc xử lý rất lâu.

Vì nhu cầu trên nên các thuật toán kiểm tra số nguyên tố như Fermat, Miller-Rabin, Solovay-Strassen, Frobenius, baillie-PSW, eliptic curve, pocklington, … ra đời để giải quyết vấn đề về tốc độ và bảo mật.

Để hiểu rõ hơn nhóm sẽ phân tích và cài đặt thuật toán kiểm tra số nguyên tố **Miller-Rabin**.

Miller-Rabin là một thuật toán xác suất để kiểm tra tính nguyên tố của một số, Được đề xuất bởi Gary L. Miller và là thuật toán dựa trên giả thiết Riemann tổng quát. Thuật toán dựa trên định lý quan trọng sau

- *Nếu n là số nguyên tố thì $(n-1)! \equiv (n-1) \pmod n$*
- *Với mỗi số nguyên tố, $Ø(n)$ là các số nguyên tố cùng nhau với n nhưng nhỏ hơn n. Khi đó, $\forall x, x > 0, x^{Ø(n)} \equiv 1 \pmod n$*

2. Phân tích

Giống với Fermat và Solovay-Strassen, Miller-Rabin sử dụng nguyên lý đồng dư (Mô-đun) nhưng sử dụng thêm hệ thống các kết quả (System of congruence) để giữ các giá trị thử nghiệm.

Với một số n, ta có thể viết như sau:

$n – 1 = 2^s * d$, d là số lẻ

ví dụ: $11 = 2^0 * 11$ $36 = 2^2 * 9$

Nếu như n là một số nguyên tố thì theo định lý fermat

$a^{n-1} \equiv 1 \bmod n$ với $n-1 = 2^s d$, d là số lẻ

$a^{n-1} \equiv 1 \pmod n \Leftrightarrow a^{2^\wedge s\ *d} - 1 \equiv 0 \bmod n$

⇔ $(a^{2^{(s-1)} * d} + 1)(a^{2^{(s-1)} * d} - 1) \equiv 0 \bmod n$

⇔ $(a^{2^{(s-1)} * d} + 1)(a^{2^{(s-2)} * d} + 1)(a^{2^{(s-2)} * d} - 1) \equiv 0 \bmod n$ ⋮

⇔ $(a^{2^{(s-1)} * d} + 1)(a^{2^{(s-2)} * d} + 1) \ldots (a^{d} + 1)(a^{d} - 1) \equiv 0 \bmod n$

(1) Khi n là số nguyên tố thì chắc chắn một trong các hệ số này phải bằng 0 mod n:

$a^d \equiv 1 \pmod{n}$ hoặc $a^{2^i * d} \equiv -1 \pmod{n}$ cho một số $i \in \{0, \ldots, s\}$

Ví dụ:

nếu $n = 13 \Rightarrow n - 1 = 2^2 * 3$

- $s = 2, d = 3$
- $a^3 \equiv 1 \pmod{n}$
- $a^3 \equiv -1 \pmod{n}$
- $a^6 \equiv -1 \pmod{n}$
- cho a từ khoảng 2 tới 12

nếu $n = 61 \Rightarrow n - 1 = 2^2 * 15$

- $s = 2, d = 15$
- $a^{15} \equiv 1 \pmod{n}$
- $a^{15} \equiv -1 \pmod{n}$
- $a^{30} \equiv -1 \pmod{n}$
- cho a từ khoảng 2 tới 60

(2) Với n lẻ > 1, $n-1 = 2^s * d$, d lẻ và chọn a trong khoảng 1 tới n-1

- Nếu toàn bộ các đồng dư trong (1) đều sai, Thì ta nói a là Miller-rabin witness

$a^d \not\equiv 1 \pmod{n}$ và $a^{2^i * d} \not\equiv -1 \pmod{n}$ cho toàn bộ i trong khoảng 0 tới n – 1

- Nếu có một đồng dư trong (1) đúng, Thì ta nói a là Miller-rabin nonwitness

$a^d \equiv 1 \pmod{n}$ và $a^{2^i * d} \equiv -1 \pmod{n}$ cho vài i trong khoảng 0 tới n – 1

Như thuật giải fermat và solovay-Strasen, Miller-rabin witness là định nghĩa để chỉ một số n là hợp số

Và một số nguyên tố thì sẽ không có Miller-rabin witness

(3) Tỉ lệ để a là witness là: ½

Thật vậy, nếu ta kiểm tra sẽ thấy 1 sẽ không là thể là witness với mọi n, ta chỉ có thể chọn a từ tập {2, …, n-1} với xác suất của 1 phần tử được chọn là 1/(n-1) => tỉ lệ chọn trúng a là một witness là $\frac{1}{n-1} * \frac{n-1}{2}$ = ½

(4) nếu n là một số hợp lẻ thì tỉ lệ để a là witness là: 75%

(5) Tỉ lệ để thuật toán kiểm tra chính xác một số nguyên tố

Ta cần kiểm tra

Pr[n prime|‘prime’] = ?

Ta đã biết:

Pr[‘composite’|n prime] = 0

Pr[‘prime’|n prime] = 1

Pr[‘composite’|n composite] = 1

Pr[‘prime’|n composite] = $(\frac{1}{4\text{^}i})$

$\frac{\Pi(n)}{2} \sim \frac{1}{\ln n}$

Pr[n prime] ~ $\frac{1}{\ln n}$

Pr[n composite] ~ $\frac{\ln n - 1}{\ln n}$

Ta có công thức Bayes Formula:

$$\Pr[A|B] = \frac{\Pr[B|A]\Pr[A]}{\Pr[B|A]\Pr[A] + \Pr[B|\overline{A}]\Pr[\overline{A}]}.$$

Với A là n prime, B là ‘prime’ ta có:

Pr[n prime|‘prime’] = $\frac{\Pr[\text{′prime′}|\text{n prime}]\ \Pr\ [\text{n prime}]}{\Pr[\text{′prime′}|\text{n prime}]\ \Pr[\text{n prime}] + \Pr[\text{′prime′}|\text{n compóite}]\ \Pr\ [\text{n composite}]}$

Pr[n prime | 'prime'] = $\frac{1*(\frac{1}{\ln n})}{1*\left(\frac{1}{\ln n}\right)+\ \frac{1}{4\hat{}i}*\left(\frac{\ln n-1}{\ln n}\right)}$

Pr[n prime | 'prime'] = $\frac{1}{1+\ \frac{1}{4\hat{}i}*(\ln n-1)}$

Nếu n ≥ $\log_4(\ln n - 1)$, thì Pr[n prime | 'prime'] ≥ 1/2

(6) Độ phức tạp thuật toán: $O(l^2 \log l)$

Vậy để thực hiện thuật toán này ta làm như sau:

+ Với n là số cần kiểm tra

+ Ta kiểm tra n có phải là số lẻ hay không, nếu không phải số lẻ thì return false

+ Phân tích n – 1 thành $2^s * d$

+ Chọn ngẫu nhiên 1 số a trong khoảng 0 tới n-1

+ Kiểm tra nếu $a^d \equiv 1 \pmod n$ thì return true

+ Nếu không thì cho i chạy từ 0 tới s

  - kiểm tra nếu $a^{i*d} \equiv -1 \pmod n$ thì return true

+ Nếu các trường hợp trên đều không đúng thì return false

Xác suất để thuật toán chạy sai: Nếu n là hợp số dương lẻ thì trong các số a ∈ {2, ..., n-1} tồn tại không quá $^{n-1}/_4$ a để n là số giả nguyên tố mạnh fermat.

3. Cài đặt

Mã giả

- Input: số tự nhiên n
- Output: True / False

```
def millerRabin(n):
  if n % 2 = 0
    return False
  a = random(2, n-1)
```

```
s, d = fermat(n - 1)

if  PowerMod(a,d,n) = 1
    return True

for i in range(0, s)
    if PowerMod(a,(2**i)*d, n) = n - 1:
        return True

return False
```

II. THỰC HÀNH

1. Ngôn ngữ: Python

2. Hướng dẫn chạy chương trình :

- Giả sử môi trường làm việc là Linux và folder bài làm được lưu trong thư mục ~/Desktop

Bước 1: Mở terminal

```
Ctrl + Shift + T
```

Bước 2: Giải nén file 19127449_19127525.zip

```
cd ~/Desktop && unzip -u ./19127447_19127525
```

Bước 3: Di chuyển vào folder 19127449_19127525

```
cd 19127447_19127525
```

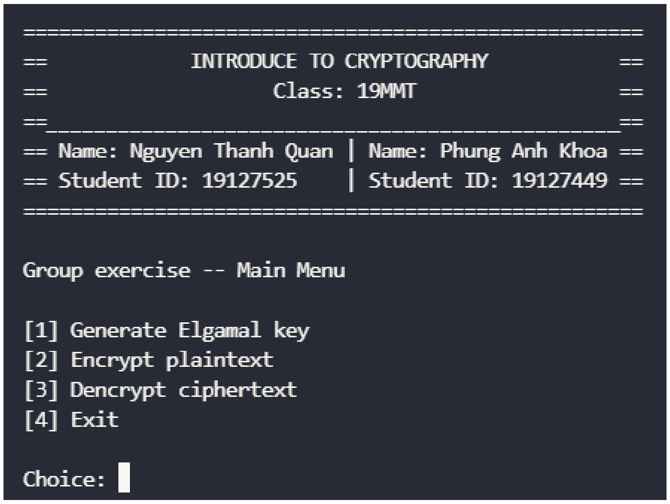
Bước 4: Chạy file bài làm

```
python3 19127449_19127525.py
```

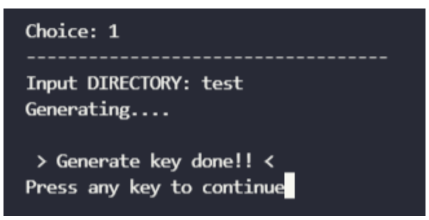
3. Hướng dẫn sử dụng:

Chạy file thực thi main.py

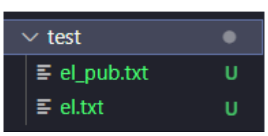
Giao diện menu chính



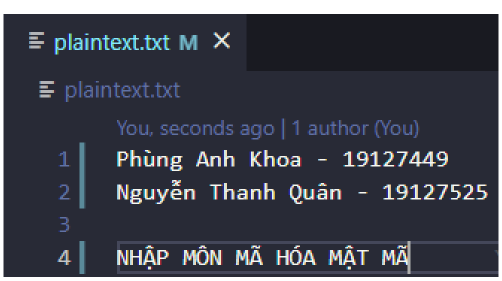
Chọn '1' để tạo Elgamal key, chương trình sẽ yêu cầu nhập folder lưu trữ file, nếu folder đã có sẵn, chương trình sẽ ghi key vào trong folder, nếu không chương trình sẽ tạo folder mới rồi ghi key vào. Ở đây cho thư mục lưu vào folder test (chưa tạo)



Đây là thư mục 'test' chương trình đã tạo tự động và ghi vào 2 file chứa khóa công khai và bí mật



Đây là nội dung file plaintext.txt (file ví dụ) để chạy demo



Chọn '2' để mã hóa file plaintext. Chương trình sẽ yêu cầu nhập vào tên file plaintext cần mã hóa và thư mục chứa khóa. Ở đây là file file plaintext là plaintext.txt và thư mục chứa khóa là test (đã được tạo ở trên)

```
Choice: 2
----------------------------------
Input PLAIN TEXT file (.txt): plaintext.txt
Input DIRECTORY: test

   > Encrypt done!! <
Press any key to continue
```

Sau khi nhập vào input, chương trình sẽ tạo file encryoted.txt và ghi nội dung đã được mã hóa của file plaintext.txt. Đây là nội dung của file encrypted.txt

plaintext.txt M   encrypted.txt M

encrypted.txt

You, seconds ago | 1 author (You)


```
NjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUx
MTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTgg
NjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUx
MTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTgg
NjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUx
MTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTgg
NjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUx
MTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTgg
NjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUx
MTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTggNjU1MTE1OCA2NTUxMTU4IDY1NTExNTgg
NjU1MTE1OCA2NTUxMTU4IDY1NTExNTgvMjEzMjM3NjUgMTA5NDM2NjEgMjY1MjQyODkgODM0ODYzNSAx
OTEzNDMgODUyOTUwNiAyNzgxMTMzMCA4MzQ4NjM1IDEwOTQzNjYxIDg1Mjk1MDYgMTExNjY0MiAxMDk0
MzY2MSAxOTEwMDk1MyAyNzg2MzY5IDg1Mjk1MDYgMTQwOTE3NzIgODUyOTUwNiAyMzU0NjU3NyA4OTAx
NzIwIDIzNTQ2NTc3IDc0NDQyOCAyMDk1MTU1MSAyMjI0OTA2NCAyMjI0OTA2NCA4OTAxNzIwIDY4NTk3
NzkgMzMzNzM1OTYgMTkxMzQzIDE2NTA1OTI3IDI1OTYwNzMyIDQ1MzQxNzggODM0ODYzNSA4NTI5NTA2
IDMwNzc4NTcwIDEwOTQzNjYxIDI3ODYzNjkgODM0ODYzNSAxMDk0MzY2MSA4NTI5NTA2IDMyMDc2MDgz
IDE2NTA1OTI3IDE0MTAyMjQ0IDgzNDg2MzUgODUyOTUwNiAxNDA5MTc3MiA4NTI5NTA2IDIzNTQ2NTc3
IDg5MDE3MjAgMjM1NDY1NzcgNzQ0NDI4IDIwOTUxNTUxIDMzMDAxMzgyIDc0NDQyOCAzMzAwMTM4MiA2
ODU5Nzc5IDY4NTk3NzkgMzMzNzM1OTYgMjQxNDE1NSA0MTYxOTY0IDIxMzIzNzY1IDg1Mjk1MDYgMjI2
MjEyNzggMzEzNDIxMjcgMzMzNzM1OTYgODUyOTUwNiAyMjYyMTI3OCAxNjMyNTA1NiA4NTI5NTA2IDI0
MTQxNTUgMjA1ODk4MDkgMjc4MTEzMzAgODUyOTUwNiAyMjYyMTI3OCA0MTYxOTY0IDMwNzc4NTcwIDg1
Mjk1MDYgMjI2MjEyNzggMTYzMjUwNTY=
```

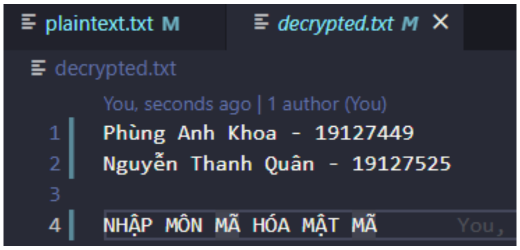

You, seconds ago • Uncommitted changes

Chọn '3' để giải mã (decrypt) file mã hóa, chương trình sẽ yêu cầu nhập vào file mã hóa (ciphertext) và thư mục chứa key. Ở đây file mã hóa là file encrypted.txt đã được mã hóa ở trên và thư mục chứa khóa là test

```
Choice: 3
----------------------------------
Input CIPHER TEXT file (.txt): encrypted.txt
Input DIRECTORY: test

    > Decrypt done!! <
Press any key to continue
```

Sau khi nhập vào input, chương trình sẽ tạo file decryoted.txt và ghi nội dung đã được giải mã của file encrypted.txt. Đây là nội dung của file decrypted.txt



4. Cấu trúc file:

```
19127449_19127525/
 |__Source/
 |   |__main.py
 |   |__miller_rabin.py
 |   |__plaintext01.txt
 |   |__plaintext02.txt
 |   |__encrypted.txt
 |   |__decrypted.txt
 |   |__QuansKey/
 |   |   |__el_pub.txt
 |   |   |__el.txt
 |__Report/
     |__report.pdf
```

File có cấu trúc như sau cây thư mục ở trên

Trong đó báo cáo là file **report.pdf** nằm ở **./19127449_19127525/Report/**

Các file mã nguồn nằm trong thư mục **Source** nằm ở **./19127449_19127525/Source** gồm có:

- file thực thi **main.py**
- file thuật toán miller rabin **miller_rabin.py**
- các file muốn encrypt/decrypt (file chứa plaintext, thực hiện nhiệm vụ 2 - 3) là file **plaintext01.txt** và **plaintext02.txt** (2 file plaintext ví dụ) nằm cùng cấp với file thực thi
- 2 file **encrypted.txt** và **decrypted.txt** là 2 file mã hóa và giải mã plaintext ở trên, nằm cùng cấp với file thực thi
- Các cặp khóa sẽ được lưu trong 2 file **el.txt** và **el_pub.txt** vào thư mục con chứa khóa, ở đây là QuansKey (folder này là ví dụ). Các folder chứa khóa này nằm cùng cấp với file thực thi

5. Giải thích source code:

3 hàm chính **generate_key(directory)**, **encryption(plaintext, alpha, beta, p)** và **decryption(ciphertext, a, p0)**

5.1 Hàm generate_key(directory):

- Input: *directory* (địa chỉ folder người dùng muốn lưu khóa)

- Output: cặp khóa sinh ra ghi vào các file *.txt* trong *directory*

- Thuật toán:

+ Bước 1: khởi tạo 2 số nguyên tố *p,* alpha và a sao cho (p > alpha > a)

+ Bước 2: tính $beta = alpha^{a} \bmod p$

+ Bước 3: ghi các khóa công khai (alpha, beta, p) vào file *el_pub.txt* và khóa bí mật (a, p) vào file *el.txt*

- Mã giả:

```
def generate_key(directory):
  # generate p
  p = generate_prime()
  do:
    alpha = generate_prime()
        while alpha >= p
  do:
    a = generate_prime()
        while a >= alpha

      # calulate beta
  beta = PowerMod(alpha, a, p)
  # save key
  open(directory + '/el_pub.txt') as file:
    file.write(alpha, beta, p)

  open(directory + '/el.txt') as file:
    file.write(a, p)
```

5.2 Hàm encryption(plaintext, alpha, beta, p):

- Input: *plaintext* lấy từ file có định dạng *.txt* do người dùng nhập vào, *alpha, beta* và *p* là khóa công khai được lấy từ file *el_pub.txt* trong folder người dùng chọn

- Output: file *encrypted.txt* (chứa thông điệp đã được mã hóa (ciphertext) từ các dữ liệu input)

- Thuật toán:

+ Bước 1: lấy k ngẫu nhiên trong khoảng (1, p)

+ Bước 2: đọc từng kí tự trong *plaintext*, chuyển chúng thành *mã ascii*. Ứng với mỗi kí tự M, sinh ra 1 tuple (c1, c2) = $(alpha^k \ mod\ p,\ (M * beta^k)\ mod\ p)$. Cho c1 vào list C1, c2 vào list C2, gộp 2 list đó vào chuỗi cách bởi dấu '/', mã hóa chuỗi đó sang base64 ta được ciphertext và gửi

. ví dụ: message='alo', alpha = 11, a=5, k = 8, p = 37

Beta = $alpha^a\ mod\ p$ =$11^5\ mod\ 37 = 27$

Ta có các cặp tuple:

. 'a'=97=(10, 6)

. 'l'=108=(10, 33)

. 'o'= 111=(10, 0)

C1 = [10, 10, 10]

C2 = [6, 33, 0]

Gộp -> message = "10 10 10/6 33 0"

Mã hóa sang base64 = 'MTAgMTAgMTAvNiAzMyAw'

+ Bước 3: mở file *encrypted.txt* và ghi ciphertext vào

- Mã giả:

```
def encryption(plaintext, n, e):
    k = randint(1, p)
    C1 = []
    C2 = []

    for message_text in plaintext:
        for i in message_text:
            M = ord(i)
```

```
        c1 = PowerMod(alpha, k, p)
        c2 = ((M % p) * PowerMod(beta, k, p)) % p

        C1.append(c1)
        C2.append(c2)


    message = str(C1) + '/' + str(C2)
    ciphertext = encode(message, base64)

    # write encrypt message into file
    with open('encrypted.txt') as file:
      file.write(ciphertext)
```

5.3 Hàm decryption(ciphertext, a, p):

- Input: *ciphertext* lấy từ file có định dạng *.txt* do người dùng nhập vào, *a* và *p* là khóa bí mật được lấy từ file *el.txt* trong folder người dùng chọn

- Output: file *decrypted.txt* (chứa thông điệp đã được giải mã (plaintext) từ các dữ liệu input)

- Thuật toán:

+ Bước 1: decode ciphertext từ base64 sang ascii

+ Bước 2: tách chuỗi đã giải mã thành 2 list C1 và C2 dựa vào dấu '/'

. ví dụ: ciphertext = "10 10 10/6 33 0"

➔ C1 = [10, 10, 10]
➔ C2 = [6, 33, 0]

+ Bước 3: ứng với mỗi phần tử của C2, ta giải mã ra kí tự ascii theo công thức $M = (C2[i] * C1[i]^{p-1-a}) mod\ p$ rồi chuyển sang dạng chữ số

. ví dụ: C1= [10, 10, 10], C2 = [6, 33, 0], a = 5, p = 37

M1 = 97

M2 = 108

M3 = 111

Chuyển sang dạng chữ số = ‘alo’ -> plaintext = ‘alo’

+ Bước 4: mở file *decrypted.txt* và ghi plaintext vào

- Mã giả

```
def decryption(ciphertext, n, d):
  plaintext = ''
  ciphertext = decode(ciphertext, base64)

  C = ciphertext.split('/')
  C1 = C[0].split(' ')
  C2 = C[1].split(' ')

  for i in range(len(C1)):
    c1 = C1[i]
    c2 = C2[i]
    M = ((c2 % p) * PowerMod(c1, p-1-a, p)) % p
        plaintext += chr(M)

   with open('decrypted.txt') as file:
      file.write(plaintext)
```

III. Ref

https://codelearn.io/sharing/thuat-toan-kiem-tra-tinh-nguyen-to

https://en.wikipedia.org/wiki/Primality_test

https://vi.wikipedia.org/wiki/Ki%E1%BB%83m_tra_Miller-Rabin

https://www.math.arizona.edu/~jtaylor/notes/crypto_talks/primality_testing_and_the_miller-rabin_algorithm.pdf

https://123docz.net/document/19067-xay-dung-chuong-trinh-kiem-tra-so-nguyen-to-bang-thuat-toan-miller-rabin-doc-doc.htm

https://kconrad.math.uconn.edu/blurbs/ugradnumthy/millerrabin.pdf

https://viblo.asia/p/ma-hoa-elgamal-hoat-dong-the-nao-bWrZnmynKxw https://en.wikipedia.org/wiki/ElGamal_encryption

https://www.slideshare.net/hoaikhong/h-mt-m-elgamal-62248712

https://boxhoidap.com/elgamal-la-gi