BÁO CÁO THỰC HÀNH

**Môn học:** **Mật mã học**

**Tên chủ đề: Public-key Cryptography**

*GVHD: Tô Trọng Nghĩa*

**Nhóm: 15 (tên nhóm: Vô Tri)**

1. **THÔNG TIN CHUNG:**

*(Liệt kê tất cả các thành viên trong nhóm)*

Lớp: NT219.N21.ANTT.2

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Họ và tên | MSSV | Email |
| 1 | Đào Võ Hữu Hiệp | 21522065 | 21522065@gm.uit.edu.vn |
| 2 | Hoàng Anh Khoa | 21522220 | 21522220@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-1)**

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Nội dung | Tình trạng | Trang |
| 1 | Yêu cầu 1 | 100% | 2 - 5 |
| 2 | Yêu cầu 2 | 100% | 6 - 9 |
| 3 | Yêu cầu 3 | 100% | 10 - 13 |
| 4 | Yêu cầu 4 | 100% | 14 - 15 |
| Điểm tự đánh giá | | | **9.5/10** |

**Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.**

BÁO CÁO CHI TIẾT

## B. LAB TASK

**1. Number Theory**

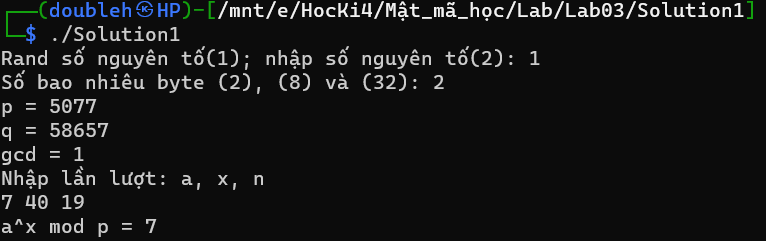
Nhóm em có đoạn code cpp như sau:

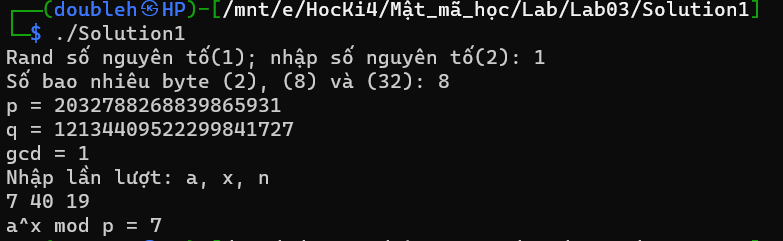
|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109 | #include <gmpxx.h>  #include <gmp.h>  #include <iostream>  **using** **namespace** std;  **void** **Rand\_2Prime**(**mpz\_t** &p, **mpz\_t** &q){  // Khởi tạo bộ sinh số ngẫu nhiên  **gmp\_randstate\_t** state;  gmp\_randinit\_mt(state);  gmp\_randseed\_ui(state, time(NULL));  // Chọn option  **int** flats;  cout << "Số bao nhiêu byte (2), (8) và (32): ";  cin >> flats;  **if**(flats == **2**){  mpz\_urandomb(p, state, **16**);  mpz\_urandomb(q, state, **16**);  }  **else** **if**(flats == **8**){  mpz\_urandomb(p, state, **64**);  mpz\_urandomb(q, state, **64**);  }    **else** **if**(flats == **32**){  mpz\_urandomb(p, state, **256**);  mpz\_urandomb(q, state, **256**);  }  **else**  exit(**1**);    mpz\_nextprime(p, p);  mpz\_nextprime(q, q);  //In 2 số p và q  cout << "p = " << mpz\_get\_str(NULL, **10**, p) << endl;  cout << "q = " << mpz\_get\_str(NULL, **10**, q) << endl;  // Giải phóng bộ nhớ  gmp\_randclear(state);  }  **void** **Input\_Prime**(**mpz\_t** &p, **mpz\_t** &q){  //Nhập 2 số  cout << "Nhập số p: ";  mpz\_inp\_str(p, stdin, **10**);  cout << "Nhập số q: ";  mpz\_inp\_str(q, stdin, **10**);  //Kiểm tra xem có phải số nguyên tố hay không. Nếu phải thì in ra.  **if**(mpz\_probab\_prime\_p(p, **0xFF**))  cout << "p = " << mpz\_get\_str(NULL, **10**, p) << endl;  **else**  cout << "Số p không phải là số nguyên tố."<< endl;  **if**(mpz\_probab\_prime\_p(q, **0xFF**))  cout << "q = " << mpz\_get\_str(NULL, **10**, q) << endl;  **else**  cout << "Số q không phải là số nguyên tố."<< endl;  }  **int** **Modular\_ax\_mod\_n**(**int** a, **int** x, **int** n){  **int** result = a % n;  **for**(**int** i = **1**; i < x; i++){  result \*= a % n;  result %= n;  }  **return** result % n;  }  **int** **main**()  {  //Khởi tạo giá trị  **mpz\_t** p, q, gcd;  mpz\_init(q);  mpz\_init(p);  mpz\_init(gcd);  //Chọn option  **int** Op;  cout << "Rand số nguyên tố(1); nhập số nguyên tố(2): ";  cin >> Op;  **if**(Op == **1**){  //Chọn ngẫu nhiên 2 số  Rand\_2Prime(p, q);  }  **else** **if**(Op == **2**){  //Nhập vào 2 số và kiểm tra xem số đó có phải số nguyên tố không  Input\_Prime(p, q);  }  **else**  exit(**1**);  //Tính UCLN  mpz\_gcd (gcd, p, q);  cout << "gcd = " << mpz\_get\_str(NULL, **10**, gcd) << endl;    //Khởi tạo và nhập 3 biến sau đó tính Modular  **int** a, x, n;  cout << "Nhập lần lượt: a, x, n **\n**";  cin >> a >> x >> n;  cout << "a^x mod p = " << Modular\_ax\_mod\_n(a, x, n);  //Giải phóng  mpz\_clear(gcd);  mpz\_clear(p);  mpz\_clear(q);  **return** **0**;  } | |

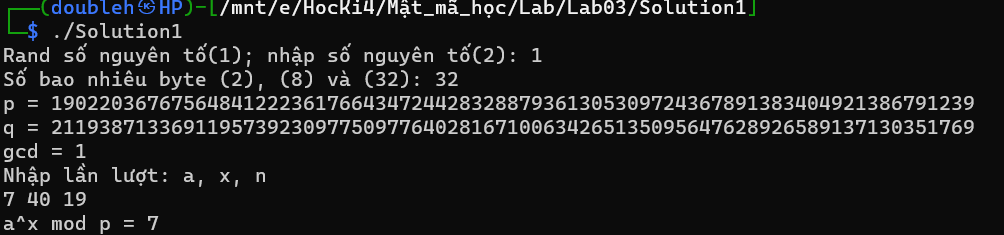
Chương trình sử dụng thư viện gmp để tính toán các số lớn. Chương trình trên có các hàm như sau:

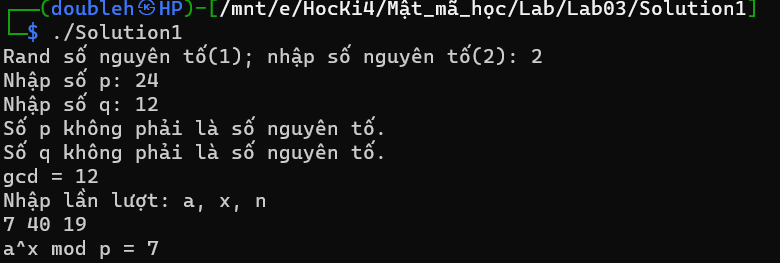
* **void** **Rand\_2Prime**(**mpz\_t** &p, **mpz\_t** &q): Chọn ngẫu nhiên 2 số nguyên tố theo các bytes tương ứng là 2, 8 và 32 bytes gán và p và q. Hàm khởi tạo bộ sinh số ngẫu nhiên dựa trên thời gian hiện tại. Hàm này sẽ in ra giá trị của p và q
* **void** **Input\_Prime**(**mpz\_t** &p, **mpz\_t** &q): Hàm này khi nhập vào 2 số bất kì nó sẽ gán vào p và q. Nếu không phải số nguyên tố thì sẽ in ra câu “Số x không phải là số nguyên tố”. Còn nếu là số nguyên tố thì sẽ in ra cả p hoặc q. Nhóm em kiểm tra số có phải số nguyên tố không bằng hàm mpz\_probab\_prime\_p(q, **0xFF**). Hàm đó của thư viện gmp sử dụng thuật toán Miller-Rabin.
* **int** **Modular\_ax\_mod\_n**(**int** a, **int** x, **int** n): Hàm này tính modular ax mod n bằng cách tách ra (a × a) ≡ [(a mod n) × (a mod n) mod n] để có thể tính các số với số mũ lớn.
* Bên cạnh đó trong hàm main nhóm em còn có dùng hàm mpz\_gcd của thư viện gmp để tính UCLN giữa 2 số.

Kết quả chạy thử:





****



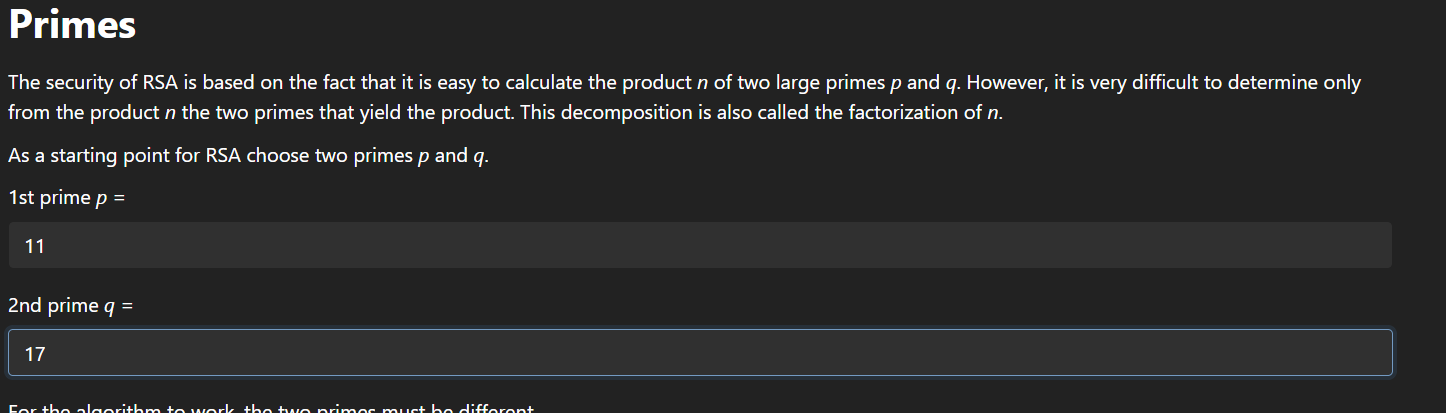
**2. RSA Public-Key Encryption**

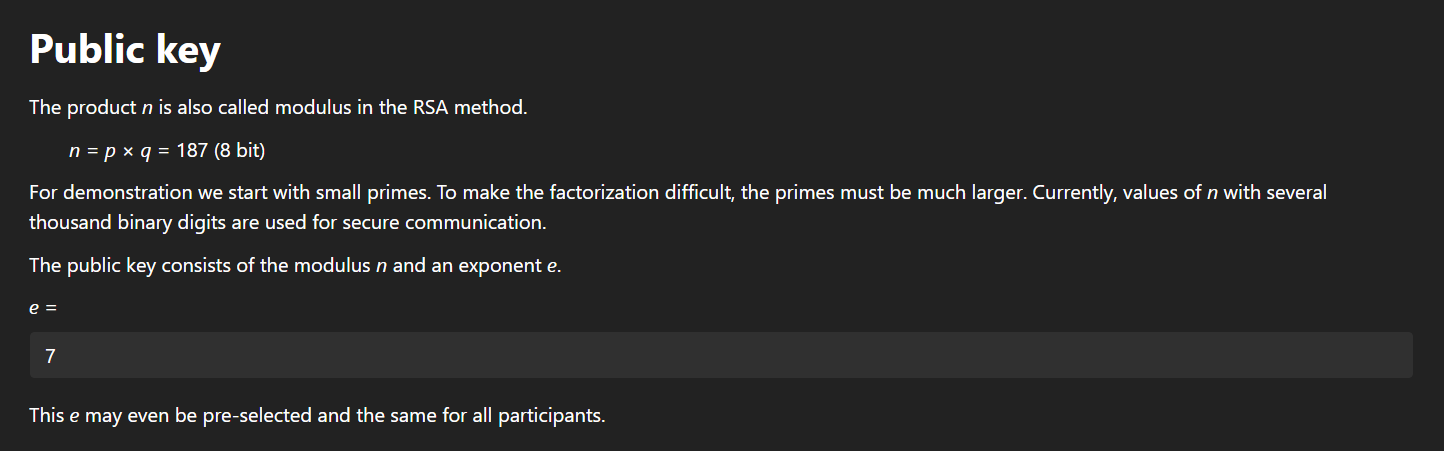
**2.1.**

* Public key PU và private key PR với p1=11 , q1=17, e1=7:

PU = (e,n) = (7 , 187)

PR = (d,n) = (23 , 187)





A screenshot of a computer

Description automatically generated with medium confidence

* Public key PU và private key PR với p2= 20079993872842322116151219 , q2= 676717145751736242170789, e2=17:

PU = (e,n) = (17 , 13588476140342208394395166469647627226674348541791)

PR = (d,n) = (7993221259024828467291262184080358019185876599873 , 13588476140342208394395166469647627226674348541791)

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

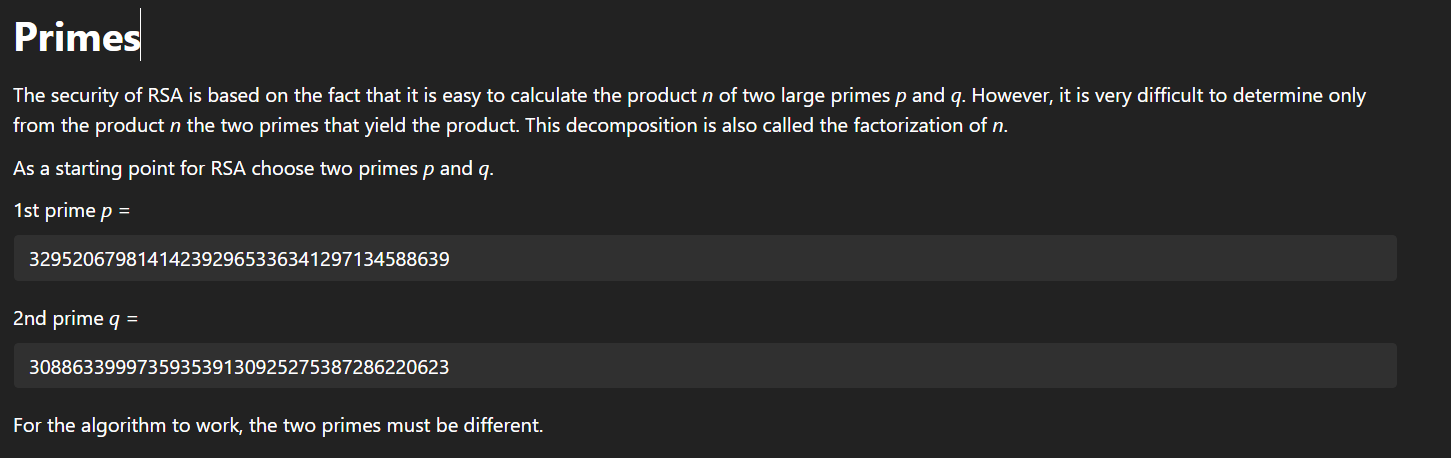
* Public key PU và private key PR với p3= F7E75FDC469067FFDC4E847C51F452DF(HEX)= 329520679814142392965336341297134588639(DEC),

q3= E85CED54AF57E53E092113E62F436F4F(HEX)= 308863399973593539130925275387286220623(DEC),

e3= 0D88C3(HEX)= 886979(DEC) :

PU = (e,n) = (886979 , 101776877529005912638346811918779931246783058062684819617574643018368103302097)

PR = (d,n) = (24212225287904763939160097464943268930139828978795606022583874367720623008491 , 101776877529005912638346811918779931246783058062684819617574643018368103302097)



A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Link tool: <https://www.cryptool.org/en/cto/rsa-step-by-step>

2.2.

* Encryption for Confidentiality với p1=11 , q1=17, e1=7: C=E(M,PU)=Me mod n

=57 mod 187

= 146

* Encryption for Authentication với p1=11 , q1=17, e1=7: C = E(M,PR) = Md mod n

= 523 mod 187

= 180

2.3.

- Tách message thành từng từ: “The “, “Faculty “, “of “, “Computer “, “Networks “, “and “, “Communications”

- Hex Encode các phần của chuỗi lần lượt: “0x54686520”, “0x466163756c747920”, “0x6f6620”, “0x436f6d707574657220”, “0x4e6574776f726b20”, “0x616e6420”, “0x436f6d6d756e69636174696f6e73”

- Mã hoá tin nhắn sử dụng RSA:

* Sử dụng bộ key đầu tiên: Vì M<n , trongkhi đó n=187(DEC)=0xbb . Vì thế M quá lớn => không thể sử dụng key đầu tiên
* Sử dụng key thứ hai: (Đổi các giá trị của p2, q2, e2 sang hex trước):

Lần lượt các phần của chuỗi sau khi encrypt: “0x26b34050ce7cc861d501f56012655b95fceb46323”, “0x3c2ffcf884200af62b64fe9b0aca1e603506b1a0b”, “0x641dd98d007e4283ec38d20628e6a340c34ab087f”, “0x65f1a9e4b9eef3f6b5d07175126f0eb3d28fee4b7”, “0x898742388b9865e1d5d89a5480170b2e66659dc1e”, “0x5f4e077920041c1605a273f6a14aacadaee13fde7”, “0x7ff947464aa7c696cafb726cb447dfbd626a9f488”

* Sử dụng key thứ ba:

Lần lượt các phần của chuỗi sau khi encrypt:  
“0xcfdeb51a62a19014fdc5b1e5df60a907959154282c1058bdefeea083eb9ecbb2”, “0x7cd47deb581dca0fceabbbb01d383b614d341e793853b643c830ec21e969a252”,

“0x7e8d88bc62a5247e463c7170fab9c7150ec5d43159608ee2b5198914d807c720”, “0x16fecd8aa9a75fc6308a4c8fc2f04a95a5f6e456d78540f0fe52a14b453a7d1a”, “0xb9b71b54fb273014e70075bb1ce60463a33b3d115179271b48332c4147ac9b33”, “0x124d5abf18275c8702323d9e29a2b6cb1ce0cf37af7d7ed79dff9358c25b271c”, “0x4f5c685ba310ab2315703c992248429c9cc6fcf76e627462bd23e9310e535679”

Tool encode text sang hex: <https://gchq.github.io/CyberChef/>

Tool RSA encryption: <https://www.tausquared.net/pages/ctf/rsa.html?fbclid=IwAR1oQVxLnClEHeojDMPnMFVRgBXlLXPzqNjREyfB8bKCSeLhXWGzWWoTIl0>

**3. RSA Application**

Code python:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134 | **import** **gmpy2**  **import** **random**  **import** **numbers**  **def** **is\_probable\_prime**(n, k=**10**):  #Kiểm tra số nguyên tố n theo thuật toán Miller-Rabin  **if** n == **2** **or** n == **3**:  **return** True  **if** n <= **1** **or** n % **2** == **0**:  **return** False  # Phân tích n-1 thành d.2^r  d = n - **1**  r = **0**  **while** d % **2** == **0**:  d //= **2**  r += **1**  # Kiểm tra tính nguyên tố của n với độ chính xác k  **for** i **in** range(k):  a = random.randint(**2**, n - **2**)  x = pow(a, d, n)  **if** x == **1** **or** x == n - **1**:  **continue**  **for** j **in** range(r - **1**):  x = pow(x, **2**, n)  **if** x == n - **1**:  **break**  **else**:  **return** False  **return** True  **def** **generate\_prime**(bits):  #Tạo số nguyên tố ngẫu nhiên với số bit được chỉ định.  **while** True:  p = random.getrandbits(bits)  **if** is\_probable\_prime(p):  **return** p  **def** **get\_n**(p, q):  **return** p\*q  **def** **get\_phi\_n**(p, q):  **return** (p-**1**) \* (q-**1**)  **def** **get\_d**(e, phi):  **for** i **in** range(**1**, phi):  d = (**1** + i \* phi) // e  **if** d \* e % phi == **1**:  **return** int(d)  **return** **0**  **def** **encrypt\_RSA**(plaintext, public\_key):  n, e = public\_key  # Chuyển chuỗi string sang int theo từng bytes  m = **0**  **for** char **in** plaintext:  m = (m << **8**) + ord(char)    # Mã hóa bằng public key  c = pow(m, e, n)  **return** c  **def** **decrypt\_RSA**(ciphertext, private\_key):  n, d = private\_key  # Giải mã bằng private key  m = pow(ciphertext, d, n)  # Chuyển từ int thành string  plaintext = ""  **while** m > **0**:  plaintext = chr(m & **0xFF**) + plaintext  m = m >> **8**  **return** plaintext  **def** **main**():  **print**("RSA!")  #Thực hiện vòng lặp liên tục nếu nhập y kết thúc khi nhập n  **while** True:  #Nhập dữ liệu  **while** True:  **try**:  #Chọn option  option = int(input("Enter random (p, q, e) (1) or enter (p, q, e) (2): "))  **if** option == **2**:  p = int(input("Enter p: "))  q = int(input("Enter q: "))  e = int(input("Enter e: "))  **if** **not** (is\_probable\_prime(p) **and** is\_probable\_prime(q) **and** is\_probable\_prime(e)):  **raise** **ValueError**("Invalid input: p, q, e must all be prime numbers")  **break**  **elif** option == **1**:  p = generate\_prime(**1024**)  q = generate\_prime(**1024**)  e = **65537** #Số e thường được sử dụng  **break**  **except** **ValueError**:  **print**("Please enter a valid number!")  **except**:  **print**("An error occurred. Please try again.")    #Public\_Key(n, e)  public\_key = (get\_n(p,q), e)  #Private\_Key(n, d)  private\_key = (get\_n(p,q), get\_d(e, get\_phi\_n(p, q)))  **print**("p = ", p)  **print**("q = ", q)  **print**("e = ", e)  **print**("n = ", get\_n(p, q))  **print**("phi = ", get\_phi\_n(p, q))  **print**("d = ", get\_d(e, get\_phi\_n(p, q)))  plaintext = input("Enter plaintext: ")    # Encrypt  ciphertext = encrypt\_RSA(plaintext, public\_key)  **print**("Encrypted message:", ciphertext)  # Decrypt  decrypted\_plaintext = decrypt\_RSA(ciphertext, private\_key)  **print**("Decrypted message:", decrypted\_plaintext)    #Chọn option lập lại hoặc không  option =input("Do you want to continue? (y/n): ")  **if** option == "y" **or** option == "Y":  **continue**  **else**:  exit()    **if** \_\_name\_\_=="\_\_main\_\_":  main() | |

Chương trình trên có các hàm như sau:

* def is\_probable\_prime : Sử dụng thuật toán Miller-Rabin để kiểm tra tính nguyên tố của 1 số với số vòng lặp để tăng độ chính xác là k = 10. Miller\_Rabin có 4 bước:

Bước 1: Kiểm tra các trường hợp đặc biệt của n.

Bước 2: Phân tích n-1 thành d.2^r.

Bước 3: Lặp lại quá trình sau k lần để kiểm tra tính nguyên tố của n:

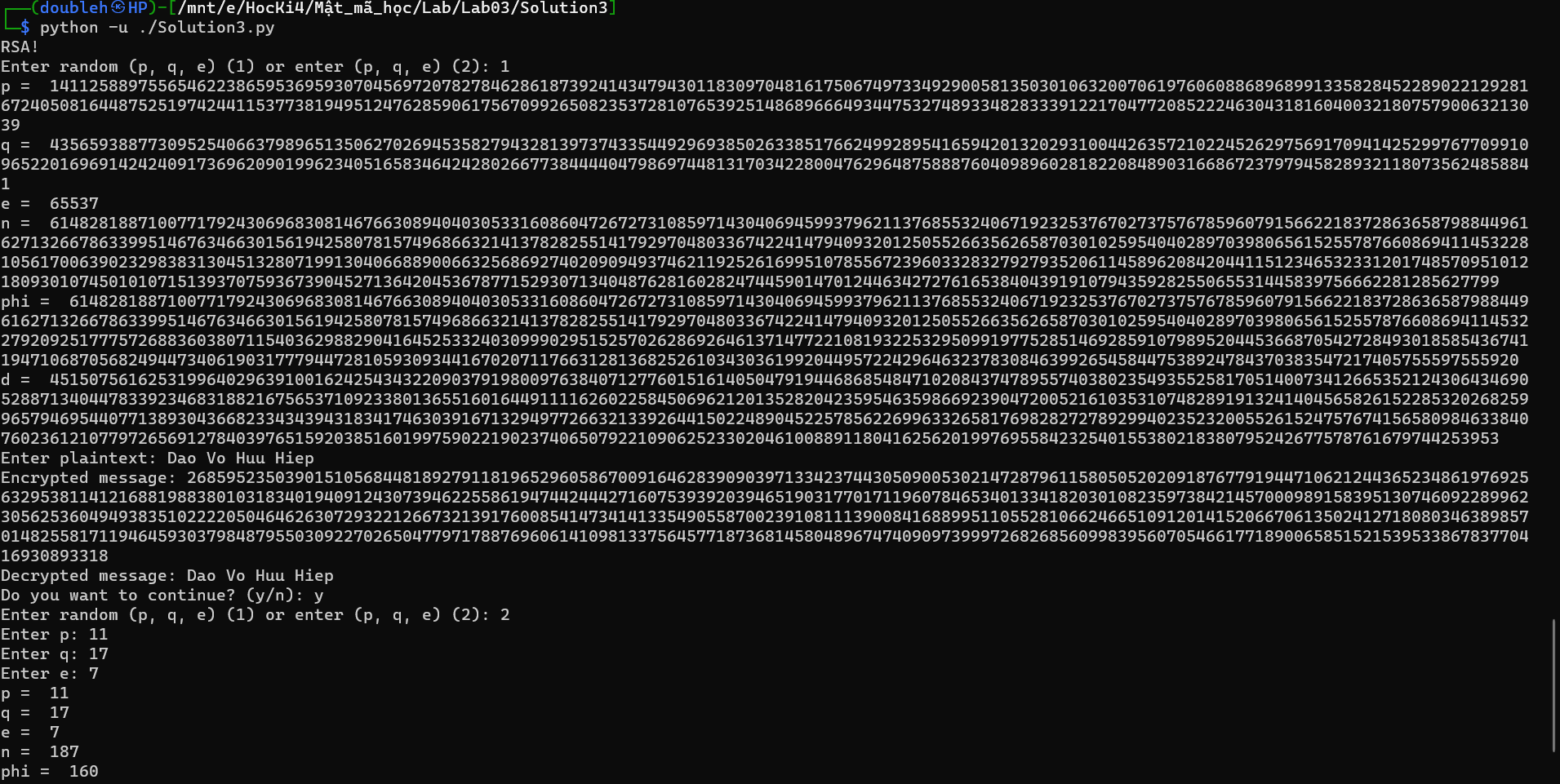
* + Chọn một số nguyên a ngẫu nhiên trong khoảng từ 2 đến n-2.
  + Tính x = a^d mod n.
  + Nếu x bằng 1 hoặc n-1, tiếp tục với lần lặp tiếp theo.
    - Lặp lại r-1 lần:
    - Tính x = x^2 mod n.
  + Nếu x bằng n-1, tiếp tục với lần lặp tiếp theo.
  + Nếu x khác n-1 ở bất kỳ lần lặp nào, trả về False.

Bước 4: Nếu chưa trả về False sau k lần lặp, trả về True.

* def generate\_prime : Tạo số ngẫu nhiên theo số bit mình dưa vào. Sau đó kiểm tra số đó có phải số nguyên tố hay không bằng hàm is\_probable\_prime
* def get\_n : Tính n = p\*q
* def get\_phi\_n : Tính phi\_n = (p-1) \* (q-1)
* def get\_d : Tính d bằng cách vét cạn theo công thức sau:

* def encrypt\_RSA : Hàm này mã hóa RSA bằng công thức sau C = Me mod n. Với bản rõ là 1 chuỗi thì em sẽ tách từng byte đổi về số theo ASCII và lưu vào 1 số nguyên M để tiến hành mã hóa
* def decrypt\_RSA : Hàm này giải mã RSA bằng công thức sau M = Cd mod n. Với sau khi giả mã ra bản rõ thì em sẽ đổi lại từ int sang chuỗi bằng cách ngược lại với các trên.

Kết quả chạy thử:

**4. Programming using the Crypto Library (required)**

Dưới đây là 3 đầu vào mẫu là 3 message tiếng Việt có dấu ngẫu nhiên có kích thước tương ứng với 50 bytes, 1000 bytes và 10000 bytes:

* Tin nhắn có kích thước 50 bytes: "Tôi yêu bạn lắm đó!"
* Tin nhắn có kích thước 1000 bytes: "Hôm nay trời nắng đẹp quá, tôi và bạn bè đã đi chơi ở công viên."
* Tin nhắn có kích thước 10 000 bytes: "Công nghệ ngày càng phát triển, đặc biệt là trong lĩnh vực truyền thông và thông tin. Các thiết bị di động ngày càng thông minh hơn, các ứng dụng truyền thông cũng ngày càng đa dạng và tiện ích hơn. Tuy nhiên, với những lợi ích đó cũng đến với những rủi ro về an ninh thông tin. Vì vậy, việc áp dụng các giải pháp bảo mật thông tin như RSA là rất cần thiết để đảm bảo an toàn cho các thông tin quan trọng."

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Message**  **đầu vào** | **Thời gian thực thi 100 lần bước tạo chữ kí cho message** | | **Thời gian thực thi 100 lần bước xác minh chữ kí cho message** | |
| **Tổng thời gian thực thi** | **Thời gian thực thi trung bình** | **Tổng thời gian thực thi** | **Thời gian thực thi trung bình** |
| 50 bytes | 46.476 ms | 0.46476 ms | 163.976 ms | 1.63976 ms |
| 1000 bytes | 51.564 ms | 0.51564 ms | 172.831 ms | 1.72831 ms |
| 10000 bytes | 55.472 ms | 0.55472 ms | 179.442 ms | 1.79442 ms |

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence A screen shot of a computer

Description automatically generated with medium confidence

A picture containing text, screenshot, font

Description automatically generated A picture containing text, screenshot, font

Description automatically generated

1. Ghi nội dung công việc, các kịch bản trong bài Thực hành [↑](#footnote-ref-1)