**BÁO CÁO BÀI TẬP LỚN CÁ NHÂN**

“ Nguyễn Đức Quốc Đại – 18020265 ”

**MỤC LỤC**

[**1. Mở đầu** 1](#_Toc73628192)

[**2. RSA** 2](#_Toc73628193)

[**3. Elgamal** 3](#_Toc73628194)

[**4. Elliptic** 5](#_Toc73628195)

# **1. Mở đầu**

- Em dùng Python để được hỗ trợ xử lý số lớn.

- 3 file chính là: RSA - Elgamal - Elliptic

+ RSA: các số được lấy 512 bit

+ Elgamal: các số được lấy 256 bit

+ Elliptic: các số được lấy 160 bit

- 2 file phụ chứa những hàm để import vào file chính là: genPrime - primitive\_root

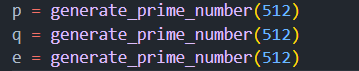
+ genPrime: File chứa hàm generate\_prime\_number(k) để sinh số nguyên tố k bit ngẫu nhiên

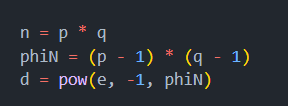
+ primitive\_root: File chứa hàm primitive(p) giúp tìm số sinh

- Để chạy Elliptic thầy phải cài thêm thư viện bằng cách:

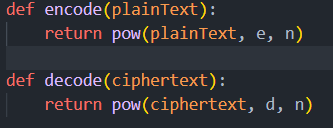
" **pip install libnum** "

# **2. RSA**





Các số q, p, e là các số nguyên tố 512 bit được sinh ngẫu nhiên bằng hàm “generate\_prime\_number” em code ở bên file “genPrime.py”. Từ đó tính được các số n, phiN, d (hàm pow(a, b, n) giúp chúng ta tính được ab mod n).



Hàm mã hóa và giải mã được tính bằng hàm pow.

Kết quả cuối cùng được thể hiện ở dưới đây, đầu vào là một số đã được chuyển hệ cơ số 10 (bất kỳ, lên đến 512 bit).

“””

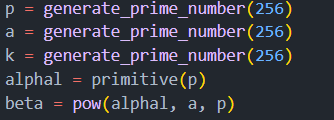
**Enter the 10-base number:** 8129471328612747123721323

**Encode:** 17538916367356375204881221456290068710886003376723040035778885421529973075968615591207115196374447766898162369089595829848817497512094309347356958686442322603904039362844843460222780152318604416691727687207682287761209324503338787620921233967390267639222479984731074445140718079574831864653732664062100482065

**Decode**: 8129471328612747123721323

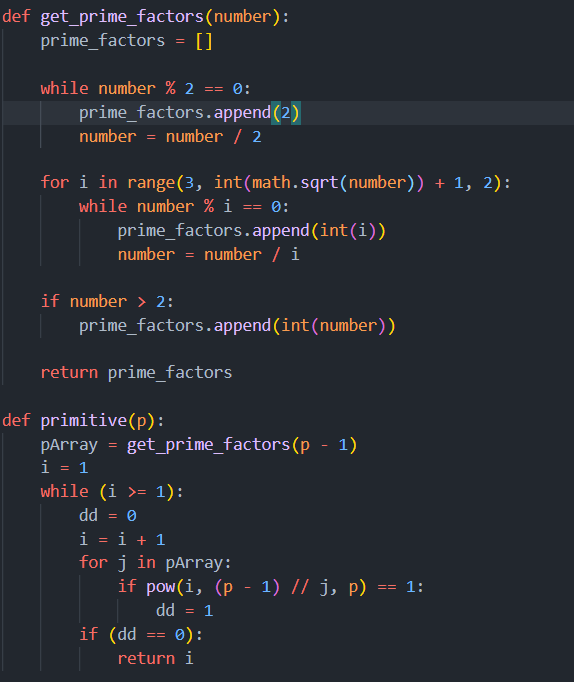
“””

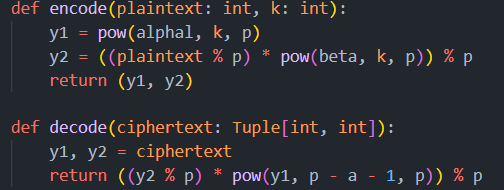
# **3. Elgamal**



Các số p, a, k là các số nguyên tố 256 bit được sinh ngẫu nhiên bằng hàm “generate\_prime\_number” em code ở bên file “genPrime.py”.

Alphal là số sinh bé nhất của số p được tính bằng hàm primitive em code ở bên file primite\_root (thuật toán tìm số sinh khá đơn giản, em tách số p-1 thành các thừa số nguyên tố p1, p2, p3, … sau đó tìm số x đầu tiên > 1 thỏa mãn x(p-1) / pi != 1 với mỗi p1, p2, p3, …). Code được thể hiện bên dưới:





Hàm mã hóa và giải mã được thực hiện theo đúng định nghĩa với sự hỗ trợ xử lý số lớn của hàm pow

Kết quả, đầu vào là 1 số cơ số 10 lên đến 256 bit.

“””

**Enter the 10-base number**: 128417498127419847194812131313

**p**: 103336558813586775588346756862650364548211208905409003711402446060831723063819

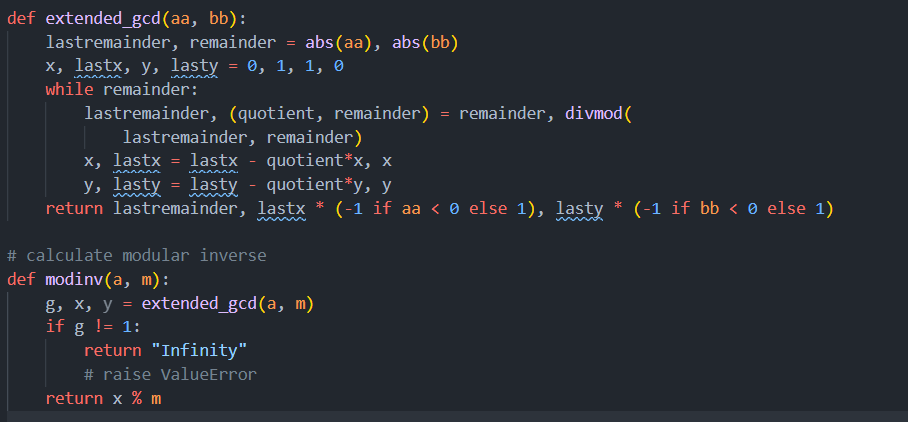
**alphal**: 2

**Encode**: (29716713186116371841931408811078771194369290659486480679277182290543483339587, 90836920171471012467551436222840460709440033047957276067636426147258800689456)

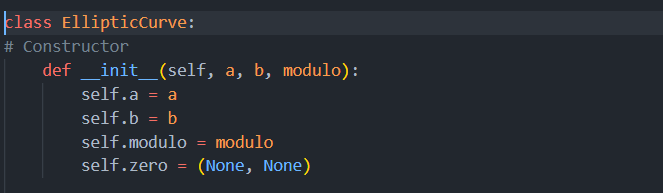
**Decode**: 128417498127419847194812131313

“””

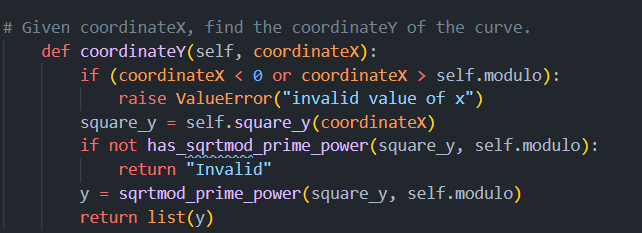
# **4. Elliptic**



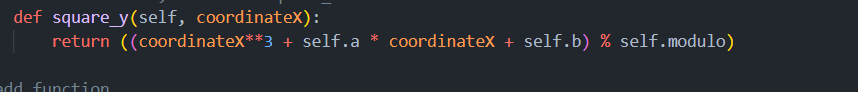
Hàm modinv dùng để tính nghịch đảo của a khi mod m, nếu ước chung lớn nhất của a, m khác 1 thì trả về Infinity. Thuật toán dùng là Euclid mở rộng.



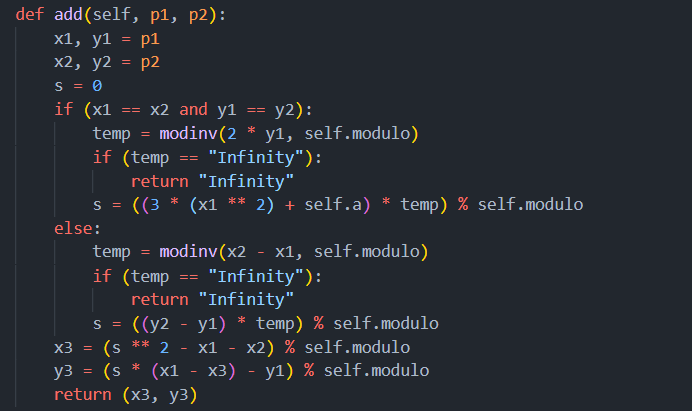
Em tạo 1 lớp EllipticCurve để thể hiện đường cong Elliptic y^2 = x^3 + a.x + b.y (mod modulo)



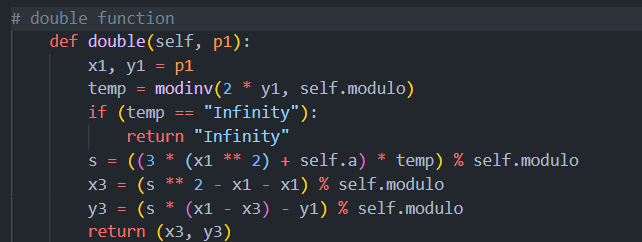
Hàm coordinateY dùng để tính tọa độ y khi biết tọa độ x



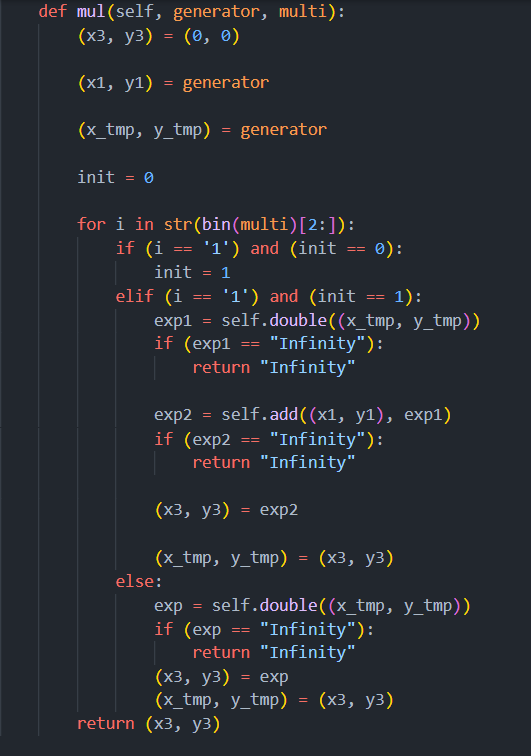
Hàm square\_y dùng để tính y^2 khi biết tọa độ x



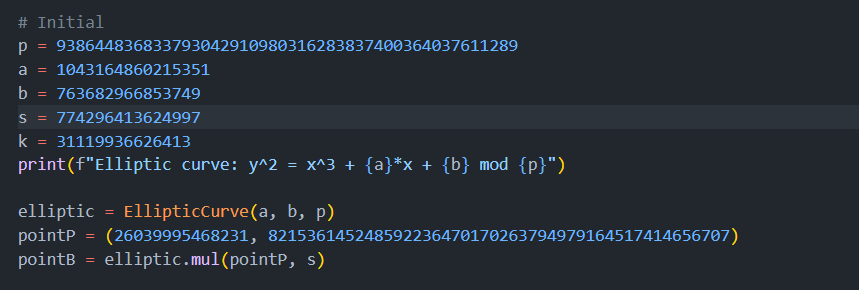
Hàm add dùng để tính phép cộng giữa 2 điểm khác nhau trên đường cong. Áp dụng đúng công thức theo quy chuẩn.



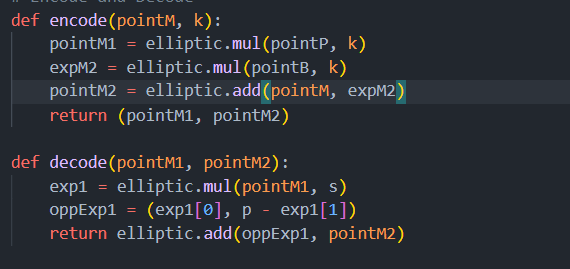
Hàm double dùng để tính phép cộng giữa 2 điểm trùng nhau trên đường cong. Áp dụng đúng công thức theo quy chuẩn.



Hàm mul dùng để tính tích của 1 điểm trên đường cong với 1 hệ số, ở đây em dùng thuật toán double and add để tính.



Các số p được lấy 160 bit, a, b, s, k được lấy 50 bit



Hàm mã hóa và giải mã được thể hiện bên trên.

Kết quả:

“””

**Elliptic** **curve**: y^2 = x^3 + 1043164860215351\*x + 763682966853749 mod 938644836833793042910980316283837400364037611289

**PlaintCode**: (33722007092201, 481027553862566585110836439308940073566227938415)

**Encode**: ((704851430803544456960203325024593024872957169188, 443833541262589641349478101172277527989715350064), (910517793822106228970938104920829400065171765477, 156604961095684834555375040889302076074099998372))

**Decode**: (33722007092201, 481027553862566585110836439308940073566227938415)

“””