BÁO CÁO BÀI TẬP LỚN CÁ NHÂN

“ Nguyễn Đức Quốc Đại – 18020265 ”

MỤC LỤC

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

```
p = generate_prime_number(512)
q = generate_prime_number(512)
e = generate_prime_number(512)
```

```
n = p * q
phiN = (p - 1) * (q - 1)
d = pow(e, -1, phiN)
```

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 a^b mod n).

```
def encode(plainText):
    return pow(plainText, e, n)

def decode(ciphertext):
    return pow(ciphertext, d, 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:

175389163673563752048812214562900687108860033767230400357788854215299730759686155912071151963744477668981623690895958298488174975120943093473569586864432260390403936284484346022278015231860441669172768720768228776120932450333878762092123396739026763922247998473107444514071807957483186465373266406210048206 5

Decode: 81294713286127471237213 23

“”””

3. Elgamal

```
p = generate_prime_number(256)
a = generate_prime_number(256)
k = generate_prime_number(256)
alphal = primitive(p)
beta = pow(alphal, a, p)
```

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:

```python
def get_prime_factors(number):
    prime_factors = []

    while number % 2 == 0:
        prime_factors.append(2)
        number = number / 2

    for i in range(3, int(math.sqrt(number)) + 1, 2):
        while number % i == 0:
            prime_factors.append(int(i))
            number = number / i

    if number > 2:
        prime_factors.append(int(number))

    return prime_factors

def primitive(p):
    pArray = get_prime_factors(p - 1)
    i = 1
    while (i >= 1):
        dd = 0
        i = i + 1
        for j in pArray:
            if pow(i, (p - 1) // j, p) == 1:
                dd = 1
        if (dd == 0):
            return i
```

```python
def encode(plaintext: int, k: int):
    y1 = pow(alphal, k, p)
    y2 = ((plaintext % p) * pow(beta, k, p)) % p
    return (y1, y2)

def decode(ciphertext: Tuple[int, int]):
    y1, y2 = ciphertext
    return ((y2 % p) * pow(y1, p - a - 1, p)) % p
```

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: 10333655881358677558834675686265036454821120890540900371140244606083172306381 9

alphal: 2

Encode:

(29716713186116371841931408811078771194369290659486480679277182290543483339587, 90836920171471012467551436222840460709440033047957276067636426147258800689456)

Decode: 128417498127419847194812131313

“””

4. Elliptic

```
def extended_gcd(aa, bb):
    lastremainder, remainder = abs(aa), abs(bb)
    x, lastx, y, lasty = 0, 1, 1, 0
    while remainder:
        lastremainder, (quotient, remainder) = remainder, divmod(
            lastremainder, remainder)
        x, lastx = lastx - quotient*x, x
        y, lasty = lasty - quotient*y, y
    return lastremainder, lastx * (-1 if aa < 0 else 1), lasty * (-1 if bb < 0 else 1)

# calculate modular inverse
def modinv(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        return "Infinity"
        # raise ValueError
    return x % m
```

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.

```python
class EllipticCurve:
# Constructor
    def __init__(self, a, b, modulo):
        self.a = a
        self.b = b
        self.modulo = modulo
        self.zero = (None, None)
```

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

```python
# Given coordinateX, find the coordinateY of the curve.
    def coordinateY(self, coordinateX):
        if (coordinateX < 0 or coordinateX > self.modulo):
            raise ValueError("invalid value of x")
        square_y = self.square_y(coordinateX)
        if not has_sqrtmod_prime_power(square_y, self.modulo):
            return "Invalid"
        y = sqrtmod_prime_power(square_y, self.modulo)
        return list(y)
```

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

```python
def square_y(self, coordinateX):
    return ((coordinateX**3 + self.a * coordinateX + self.b) % self.modulo)

dd function
```

Hàm square_y dùng để tính y^2 khi biết tọa độ x

```
def add(self, p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    s = 0
    if (x1 == x2 and y1 == y2):
        temp = modinv(2 * y1, self.modulo)
        if (temp == "Infinity"):
            return "Infinity"
        s = ((3 * (x1 ** 2) + self.a) * temp) % self.modulo
    else:
        temp = modinv(x2 - x1, self.modulo)
        if (temp == "Infinity"):
            return "Infinity"
        s = ((y2 - y1) * temp) % self.modulo
    x3 = (s ** 2 - x1 - x2) % self.modulo
    y3 = (s * (x1 - x3) - y1) % self.modulo
    return (x3, y3)
```

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

```
# double function
    def double(self, p1):
        x1, y1 = p1
        temp = modinv(2 * y1, self.modulo)
        if (temp == "Infinity"):
            return "Infinity"
        s = ((3 * (x1 ** 2) + self.a) * temp) % self.modulo
        x3 = (s ** 2 - x1 - x1) % self.modulo
        y3 = (s * (x1 - x3) - y1) % self.modulo
        return (x3, y3)
```

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

```python
def mul(self, generator, multi):
    (x3, y3) = (0, 0)

    (x1, y1) = generator

    (x_tmp, y_tmp) = generator

    init = 0

    for i in str(bin(multi)[2:]):
        if (i == '1') and (init == 0):
            init = 1
        elif (i == '1') and (init == 1):
            exp1 = self.double((x_tmp, y_tmp))
            if (exp1 == "Infinity"):
                return "Infinity"

            exp2 = self.add((x1, y1), exp1)
            if (exp2 == "Infinity"):
                return "Infinity"

            (x3, y3) = exp2

            (x_tmp, y_tmp) = (x3, y3)
        else:
            exp = self.double((x_tmp, y_tmp))
            if (exp == "Infinity"):
                return "Infinity"
            (x3, y3) = exp
            (x_tmp, y_tmp) = (x3, y3)
    return (x3, y3)
```

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.

```
# Initial
p = 938644836833793042910980316283837400364037611289
a = 1043164860215351
b = 763682966853749
s = 774296413624997
k = 31119936626413
print(f"Elliptic curve: y^2 = x^3 + {a}*x + {b} mod {p}")

elliptic = EllipticCurve(a, b, p)
pointP = (26039995468231, 821536145248592236470170263794979164517414656707)
pointB = elliptic.mul(pointP, s)
```

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

```
def encode(pointM, k):
    pointM1 = elliptic.mul(pointP, k)
    expM2 = elliptic.mul(pointB, k)
    pointM2 = elliptic.add(pointM, expM2)
    return (pointM1, pointM2)

def decode(pointM1, pointM2):
    exp1 = elliptic.mul(pointM1, s)
    oppExp1 = (exp1[0], p - exp1[1])
    return elliptic.add(oppExp1, pointM2)
```

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),

(91051779382210622897093810492082940006517176547 7,

15660496109568483455537504088930207607409999837 2))

Decode: (33722007092201, 48102755386256658511083643930894007356622793841 5)

“””