Bài tập lớn môn Nhập môn An toàn thông tin

Hoàng Đức Minh - 18020890

1. Hệ mật RSA với p,q - 512 bit

1.1. Quy trình

1.1.1. Chuẩn bị (A và B cùng thực hiện)

- **Bước 1**: Chọn 2 số nguyên tố p, q có độ lớn 512 bit bằng hàm generate_prime_number(512)
    - Từ đó, tính được n = pq và $\Phi(n) = (p-1)(q-1)$
- **Bước 2**: Chọn giá trị b ($1<b<\Phi(n)$) sao cho $gcd(b,\Phi(n)) = 1$. Để đơn giản, chọn b là số nguyên tố bằng hàm generate_prime_number(10) thu được: b=673
- **Bước 3**: Tính được $a = b^{-1} \bmod \Phi(n)$ bằng hàm modinv(b, $\Phi(n)$). Khi đó ta thu được, khóa công khai (n, b) và khóa bí mật (p, q, a)
- **Bước 4:** A và B gửi khóa công khai cho nhau

1.1.2. B mã hóa rồi gửi cho A bản mã và chữ ký

- **Bước 5**: Yêu cầu B nhập nội dung cần mã hóa và thực hiện mã hóa bằng hàm rsa_encrypt(x, n, b) với khóa công khai của A
- **Bước 6:** Tạo chữ ký bằng kết quả mã hóa bằng khóa riêng của B
- **Bước 7:** Gửi bản mã hóa và chữ ký cho A

1.1.3. A giải mã thu được bản rõ

- **Bước 8**: Xác thực chữ ký dựa trên bản mã hóa và khóa công khai của B
- **Bước 9**: Giải mã bằng hàm rsa_decrypt(encrypt, a, n)

1.2. Thiết kế thuật toán:

1.2.1. Thuật toán tính nghịch đảo modulo

Thuật toán tính nghịch đảo modulo được xây dựng dựa trên thuật toán euclid mở rộng. Với phương trình a*x+m*y=1, từ thuật toán euclid mở rộng ta có thể xác định được x và y. Trong toán học modulo với mod m, ta có thể bỏ được m*y đi, từ đó x chính là a^{-1}

```
INPUT: a, x , n ∈ Z_n
OUTPUT: b = a^x mod n
Square − multiply(a, x, n)
{   b = 1;
    for( i = 0  to  m − 1)       // m is the number of bits in x

{  if (xi = 1   then        y = y × a mod n;
                               a = a^2 mod n;
}
return b;
}
```

Thuật toán euclid mở rộng

```python
def extended_gcd(aa, bb):
    """extended euclid algorithm

    Args:
        aa : [description]
        bb : [description]

    """
    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)
```

Cài đặt thuật toán euclid mở rộng

```python
def modinv(a, m):
    """modular inverse a^-1 mod m

    Args:
        a: input number
        m : prime

    """
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise ValueError
    return x % m
```

Thuật toán nghịch đảo modulo

1.2.2. Thuật toán lũy thừa modulo

ALGORITHM 5 Modular Exponentiation.

procedure *modular exponentiation*(b: integer, $n = (a_{k-1}a_{k-2}\ldots a_1a_0)_2$,
m: positive integers)
$x := 1$
$power := b$ **mod** m
for $i := 0$ **to** $k - 1$
  if $a_i = 1$ **then** $x := (x \cdot power)$ **mod** m
  $power := (power \cdot power)$ **mod** m
return $x\{x$ equals b^n **mod** $m\}$

Thuật toán tính lũy thừa modulo

```python
def modexp(x, y, p):
    """modular exponential x^y mod p

    Returns:
        [type]: [description]
    """
    res = 1     # Initialize result
    # Update x if it is more than or equal to p
    x = x % p
    if (x == 0):
        return 0

    while (y > 0):
        if ((y & 1) == 1):
            res = (res * x) % p

        y = y >> 1
        x = (x * x) % p

    return res
```

Cài đạt thuật toán tính lũy thừa modulo

1.2.3. Cài đặt hệ mật RSA

RSA gồm hai phần mã hóa và giải mã sử dụng thuật toán tính lũy thừa modulo

Cryptosystem 6.1: *RSA Cryptosystem*

Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\phi(n)}\}.$$

For $K = (n, p, q, a, b)$, define

$$e_K(x) = x^b \bmod n$$

and

$$d_K(y) = y^a \bmod n$$

($x, y \in \mathbb{Z}_n$). The values n and b comprise the public key, and the values p, q, and a form the private key.

Mã hóa và giải mã hệ mật RSA

```python
def rsa_encrypt(x, b, n):
    """RSA encryption ek = x^b mod n

    Args:
        x : content
        b : random number that gcd(b, phi(n)) = 1
        n : pq
    """
    return modexp(x, b, n)
```

Cài đặt thuật toán mã hóa RSA

```python
def rsa_decrypt(y, a, n):
    """RSA encryption dk = y^a mod p

    Args:
        y : encrypted content
        a : modular inverse of b
        n : pq
    """
    return modexp(y, a, n)
```

Cài đặt thuật toán giải mã RSA

1.2.4. Cài đặt chữ ký hệ mật RSA

Cryptosystem 8.1: *RSA Signature Scheme*

Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) : n = pq, \text{where } p \text{ and } q \text{ are prime}, ab \equiv 1 \pmod{\phi(n)}\}.$$

The values n and b are the public key, and the values p, q, and a are the private key.

For $K = (n, p, q, a, b)$, define

$$\mathbf{sig}_K(x) = x^a \bmod n$$

and

$$\mathbf{ver}_K(x, y) = true \Leftrightarrow x \equiv y^b \pmod{n},$$

for $x, y \in \mathbb{Z}_n$.

Thuật toán xác thực RSA

```python
def rsa_signature(x, a, n):
    """RSA signature sigk = x^a mod p

    Args:
        x ([type]): [description]
        a ([type]): [description]
        n ([type]): [description]

    Returns:
        [type]: [description]
    """
    return modexp(x, a, n)
```

Tạo chữ ký RSA

```python
def rsa_verify(x, y, b, n):
    """RSA verification verification

    Args:
        x ([type]): [description]
        y ([type]): [description]
        b ([type]): [description]
        n ([type]): [description]

    Returns:
        [type]: [description]
    """
    return x == modexp(y, b, n)
```

Xác thực chữ ký RSA

1.3. Kết quả:

```
✘ chuis@chuis-HP-EliteBook-8570w ~/Desktop/Test Project/ATTT  master ±+ >M< python3 rsa.py
Public key of Alice is (n,b): ( 875189261219375547868172997669864984696646562946145960159473747730600728138243994013364757599318404907001003884381521854241382269054111019864957998205486190696511844136552289006675121349444802633533872007033432546660307555118266731826627377909022322702210214441096640435960938826766969082808595674867125607 09 , 607 )
Private key of Alice is (p,q,a): ( 81270445725214541494567013384378298776963271495860519378295017436463224594815627456917457687107996855827367340368152310509673268385193801782029751130187381 , 10768850267888267849621565075082852704729213646526654319918481430163992796570222242585546212347902009223001369452786507814821377952397377584125051830559889 , 553661740211268880364709112199716893119460099129028086822467741562686457339515146130658433577236696611011293107125278287128349115597671031765137912865018528290351910627736450049756485646881181172987640492731603717663437719189970118073639675719450762207692404110962067913453615321854691559633191181311482102 )
--------
Public key of Bob is (n,b): ( 89036297494366081489123842102450997778261681670134433530663829855262656370904036293652006405645639523287842224652385174880943847250521805994766489811049899134222416554351130582433153022367326355911543744859961147635734817091941337390374472331357632396109761690296497163642353048265600599717448406467152494429 , 967 )
Private key of Bob is (p,q,a): ( 93664980367795700005306687677898130946626637122641830727080762087932937920215857231304683691378716136591911642667112634310989388067070679610716925128660 33 , 95058256719582816090381636537963656890549979872263669010558241290656186602584071160650352017409203509163544459076229346632757231242135537987485497094920 13 , 58467475603849495083343991453005567310440711334576386030994345354800193170138638103897646398743517163689534449487347038313753198556443998765953175832488804515310940499976417625270387240930428723152689867101180960363988695391908533037678284163024393524955732370088743533295867898385036449044221563860217824823 )
--------
Alice want to send message to Bob
Enter message to encrypt:hoangducminh
Alice send to Bob code message: 40937401545520469791643600477530080681917445575974040122920493092920394524173291926215606477967634580266208701870727498492003970897877787184854869885476020545010764846297462644182786795856250866898153039684246516847757930328132475355775649340668229762748183411025577782461614148194312586487211369914363804019
--------
Bob received message
Verify: True
Message: hoangducminh
```

2. Hệ mật ElGama với p - 256 bit

2.1. Quy trình

2.1.1. Chuẩn bị (A và B cùng thực hiện)

- **Bước 1**: Chọn số nguyên tố p có độ lớn 256 bit bằng hàm generate_prime_number(256)
- **Bước 2**: Tìm phần tử nguyên thủy α bằng hàm primitive(p)
- **Bước 3**: Nhập khóa bí mật a. Từ đó tính được $\beta = \alpha^a \pmod p$. Khi đó ta thu được khóa công khai (p, α, β) và khóa mật (a)
- **Bước 4:** Sinh tự động số k bất kì (bí mật) bằng hàm k = randrange(0, 10000000, 1)
- **Bước 5:** A và B gửi khóa công khai cho nhau

2.1.2. B mã hóa dựa gửi cho A bản mã và chữ ký

- **Bước 6**: Yêu cầu B nhập nội dung cần mã hóa và thực hiện mã hóa bằng hàm elgama_encrypt(x, α, β, p) với khóa công khai của A thu được (y1, y2)
- **Bước 7**: Tạo chữ ký bằng khóa bí mật của B và gửi bản mã và chữ ký cho A

2.1.3. A giải mã thu được bản rõ

- **Bước 8**: Xác thực chữ ký bằng khóa công khai của B
- **Bước 9**: Giải mã bằng hàm elgama_decrypt(y1, y2, a, p)

2.2. Thiết kế thuật toán:

2.2.1. Thuật toán tính lũy thừa modulo

Đã trình bày ở Hệ mật RSA

2.2.2. Thuật toán xác định phần tử nguyên thủy modulo n

THEOREM 6.8 *Suppose that $p > 2$ is prime and $\alpha \in \mathbb{Z}_p^*$. Then α is a primitive element modulo p if and only if $\alpha^{(p-1)/q} \not\equiv 1 \pmod p$ for all primes q such that $q \mid (p-1)$.*

Thuật toán xác định phần tử nguyên thủy modulo

```python
def primitive(n):
    s = set()

    # Find value of Euler Totient function of n. Since n is a prime number, the
    # value of Euler Totient function is n-1 as there are n-1 relatively prime numbers.
    phi = n - 1
    # Find prime factors of phi and store in a set
    findPrimefactors(s, phi)

    # Check for every number from 2 to phi
    for r in range(2, phi + 1):

        # Iterate through all prime factors of phi and check if we found a power with value 1
        flag = False
        for it in s:

            # Check if r^((phi)/primefactors) mod n is 1 or not
            if (modexp(r, phi // it, n) == 1):

                flag = True
                break

        # If there was no power with value 1.
        if (flag == False):
            return r

    # If no primitive root found
    return -1
```

Cài đặt thuật toán xác định phần tử nguyên thủy modulo

2.2.3. Cài đặt hệ mật ElGama

ElGama gồm hai phần mã hóa và giải mã

Cryptosystem 7.1: *ElGamal Public-key Cryptosystem in* $\mathbb{Z}_p^*$

Let p be a prime such that the **Discrete Logarithm** problem in $(\mathbb{Z}_p^*, \cdot)$ is infeasible, and let $\alpha \in \mathbb{Z}_p^*$ be a primitive element. Let $\mathcal{P} = \mathbb{Z}_p^*$, $\mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and define

$$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \pmod{p}\}.$$

The values p, α, and β are the public key, and a is the private key.

For $K = (p, \alpha, a, \beta)$, and for a (secret) random number $k \in \mathbb{Z}_{p-1}$, define

$$e_K(x, k) = (y_1, y_2),$$

where

$$y_1 = \alpha^k \bmod p$$

and

$$y_2 = x\beta^k \bmod p.$$

For $y_1, y_2 \in \mathbb{Z}_p^*$, define

$$d_K(y_1, y_2) = y_2 (y_1{}^a)^{-1} \bmod p.$$

Hệ mật ElGama

```python
def elgama_encrypt(x, alpha, beta, p):
    """Elgama encryption

    Args:
        x : content
        alpha : primitive element of p
        beta : random element of p
        p : random big prime
    """
    k = randrange(0, 10000000, 1)
    y1 = modexp(alpha, k, p)
    y2 = (x * modexp(beta, k, p)) % p

    return y1, y2
```

Mã hóa ElGama

```python
def elgama_decrypt(y1, y2, a, p):
    """Elgama dectryption

    Args:
        y1 :
        y2 :
        a : discrete logarithm of beta
        p : random big prime
    """

    return (y2 * modinv(modexp(y1, a, p), p)) % p
```

Giải mã ElGama

2.2.4. Cài đặt chữ ký hệ mật RSA

Cryptosystem 8.2: *ElGamal Signature Scheme*

Let p be a prime such that the discrete log problem in $\mathbb{Z}_p$ is intractable, and let $\alpha \in \mathbb{Z}_p{}^*$ be a primitive element. Let $\mathcal{P} = \mathbb{Z}_p{}^*$, $\mathcal{A} = \mathbb{Z}_p{}^* \times \mathbb{Z}_{p-1}$, and define

$$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \pmod{p}\}.$$

The values p, α, and β are the public key, and a is the private key.

For $K = (p, \alpha, a, \beta)$, and for a (secret) random number $k \in \mathbb{Z}_{p-1}{}^*$, define

$$\mathbf{sig}_K(x, k) = (\gamma, \delta),$$

where

$$\gamma = \alpha^k \bmod p$$

and

$$\delta = (x - a\gamma)k^{-1} \bmod (p-1).$$

For $x, \gamma \in \mathbb{Z}_p{}^*$ and $\delta \in \mathbb{Z}_{p-1}$, define

$$\mathbf{ver}_K(x, (\gamma, \delta)) = true \Leftrightarrow \beta^\gamma \gamma^\delta \equiv \alpha^x \pmod{p}.$$

Chữ ký hệ mật ElGama

```python
def elgama_signature(y, a, alpha, p):
    """ElGama signature

    Args:
        x : context
        a : discrete logarithm of beta
        alpha : primitive root of p
        p : random big prime

    Returns:
        (gama, delta)
    """
    y1, y2 = y

    k = generate_prime_number(20)
    gama = modexp(alpha, k, p)

    delta1 = ((y1 - a * gama) * modinv(k, p - 1)) % (p - 1)

    delta2 = ((y2 - a * gama) * modinv(k, p - 1)) % (p - 1)
    return (gama, delta1, delta2)
```

Thuật toán sinh chữ ký hệ mật ElGama

```python
def elgama_verify(x, alpha, beta, gama, delta, p):
    """ElGama verification

    Args:
        x : context
        alpha : primitive root of p
        beta :
        gama :
        delta :
        p : random big prime

    Returns:
        Boolean
    """
    return modexp(alpha, x, p) == (modexp(beta, gama, p) * modexp(gama, delta, p)) % p
```

Thuật toán xác minh hệ mật ElGama

2.3. Kết quả:

```
chuis@chuis-HP-EliteBook-8570w ~/Desktop/Test Project/ATTT master ±+ >M< python3 elgama.py
Public key of Alice is (p, alpha, beta): 725187363839498961338318403387912331945297130819902653300433193203809998125 87 , 3 , 6828378060435538212581385053078183717252690164188961943301393811477328344622 9
Private key of Alice is (a): 4445
--------
Public key of Bob is (p, alpha, beta): 95349613533861734992301767245977480991375186565835080412744423954045491915421 , 2 , 33984672208497406318582943241839841914127772477464554562456813609234106384930
Private key of Bob is (a): 3627
--------
Alice want to send message to Bob
Enter message to encrypt:hoangducminh
Alice send to Bob coded message: (635005865810906955201517081962914015045834395891307108982967078011095677831 38, 53159433703267242802349454149470990798205649212467237945764329079613637841 87, 5720117834588157158698821312792148436458545037168731360376473642427029870062 6)
---------
Bob received message
Verify: True
Message: hoangducminh
```

3. Mã hoá trên đường cong Elliptic với p = 160 bit

3.1. Quy trình

3.1.1. Chuẩn bị (A và B cùng thực hiện)

- **Bước 1**: Chọn a=53, b=7 và p=1114597119506223026265579259036275469126397408411. Việc chọn a, b, p thông qua việc chọn p là số nguyên tố 160 bit qua hàm generate_prime_number(160), sau đó random a, b cho đến khi bậc của đường cong Elliptic là số nguyên tố. Bậc được tính bằng thuật toán schoof, điều này nhằm đảm bảo tất cả các điểm đều là điểm sinh. n=1114597119506223026265580072500994466839748057413
- **Bước 2**: Chọn điểm đầu tiên P của đường Elliptic làm điểm sinh qua hàm findFirstPoint(a, b, p)
- **Bước 3**: Nhập khóa bí mật s. Từ đó tính được B=sP. Khi đó ta thu được khóa công khai (E, p, P, B) và khóa mật (s)
- Bước 4: A và B trao đổi khóa công khai

3.1.2. B mã hóa rồi gửi cho A bản mã và chữ ký

- **Bước 5:** Yêu cầu B nhập điểm M cần mã hóa và thực hiện mã hóa bằng hàm ecc_encrypt(M, a, b, p, P, B) dựa vào khóa công khai của A thu được (M1, M2)
- **Bước 6:** B tạo chữ ký từ bản mã và khóa riêng của B rồi gửi bản mã và chữ ký cho A

3.1.3. A giải mã thu được bản rõ

- **Bước 7:** A xác minh chữ ký của B dựa vào khóa công khai của B
- **Bước 8**: AGiải mã bằng hàm ecc_decrypt(M1, M2, s, p, a)

3.2. Thiết kế thuật toán:

3.2.1. Thuật toán chọn đường Elliptic thỏa mãn

Với p đã chọn, thực hiện random a và b để tìm được đường Elliptic có bậc là số nguyên tố, từ đó tất cả các phần tử đều là phần tử sinh.

```
M = 1114597119506223026265579259036275469126397408411
ord = -1
step = 0
A = 106
B = 12
while True:
    A += 1
    if A % 10 == 0:
        B += 1
    # A = randrange(1, 100, 1)
    # B = randrange(1, 100, 1)
    F = FiniteField(M)
    E = EllipticCurve(F, [A, B])

    ord = E.order()
    step += 1
    if step % 10 == 0:
        print(step, "tries")
    if ord.is_prime():
        print("A:", A)
        print("B:", B)
        print("M:", M)
        print("Order of group:", ord)
        break
```

Thuật toán xác định đường cong Elliptic

3.2.1. Thuật toán tìm điểm thuộc đường cong Elliptic

Việc xác định điểm đầu tiên thuộc đường cong Elliptic được thực hiện bằng cách chạy x rồi tính y cho đến khi y có giá trị là phần tử thặng dư bậc 2 của p

```
def findFirstPoint(a, b, p):
    x = 0
    while True:
        f = x**3 + a*x + b
        if is_square(f):
            y = int(sqrt(f))
            return (x, y)
        else:
            x += 1
```

Thuật toán tìm điểm thuộc đường cong Elliptic

3.2.2. Thuật toán cộng trên đường cong Elliptic

(iii)Cho $P = (x_1, y_1)$ thuộc $E_p(a, b)$ và $Q = (x_2, y_2)$ thuộc $E_p(a, b)$, ở đây, $P \neq -Q$. Khi đó, $P + Q = (x_3, y_3)$, ở đây:

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

Trong đó:

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, khi\ P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1}, khi\ P = Q \end{cases}$$

Thuật toán cộng trên đường cong Elliptic

```
def ecc_add(x1, y1, x2, y2, p, a):
    s = 0
    if (x1 == x2):
        s = ((3*(x1**2) + a) * modinv(2*y1, p)) % p
    else:
        s = ((y2-y1) * modinv(x2-x1, p)) % p
    x3 = (s**2 - x1 - x2) % p
    y3 = (s*(x1 - x3) - y1) % p
    return (x3, y3)
```

Cài đặt thuật toán cộng trên đường cong Elliptic

3.2.3. Thuật toán nhân đôi trên đường cong Elliptic

```
def ecc_double(x1, y1, p, a):
    s = ((3*(x1**2) + a) * modinv(2*y1, p)) % p
    x3 = (s**2 - x1 - x1) % p
    y3 = (s*(x1-x3) - y1) % p
    return (x3, y3)
```

Cài đặt thuật toán cộng trên đường cong Elliptic

3.2.4. Thuật toán nhân và cộng trên đường cong Elliptic

Algorithm 7.6: DOUBLE-AND-(ADD OR SUBTRACT)$(P, (c_{\ell-1}, \ldots, c_0))$

$Q \leftarrow \mathcal{O}$
for $i \leftarrow \ell - 1$ **downto** 0
do
- $Q \leftarrow 2Q$
- **if** $c_i = 1$
  - **then** $Q \leftarrow Q + P$
- **else if** $c_i = -1$
  - **then** $Q \leftarrow Q - P$

return (Q)

Thuật toán nhân và cộng trên đường công Elliptic

```python
def double_and_add(multi, generator, p, a):
    (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):
            (x3, y3) = ecc_double(x_tmp, y_tmp, p, a)
            (x3, y3) y_tmp: int 1, y1, x3, y3, p, a)
            (x_tmp, y_tmp) = (x3, y3)
        else:
            (x3, y3) = ecc_double(x_tmp, y_tmp, p, a)
            (x_tmp, y_tmp) = (x3, y3)
    return (x3, y3)
```

Cài đặt thuật toán nhân và cộng trên đường công Elliptic

3.2.5. Cài đặt hệ mật ElGama Elliptic Curve

Cryptosystem 7.2: *Elliptic Curve ElGamal*

Let $\mathcal{E}$ be an elliptic curve defined over $\mathbb{Z}_p$ (where $p > 3$ is prime) such that $\mathcal{E}$ contains a cyclic subgroup $H = \langle P \rangle$ of prime order n in which the **Discrete Logarithm** problem is infeasible. Let $h : \mathcal{E} \to \mathbb{Z}_p$ be a secure hash function. Let $\mathcal{P} = \mathbb{Z}_p$ and $\mathcal{C} = (\mathbb{Z}_p \times \mathbb{Z}_2) \times \mathbb{Z}_p$. Define

$$\mathcal{K} = \{(\mathcal{E}, P, m, Q, n, h) : Q = mP\},$$

where P and Q are points on $\mathcal{E}$ and $m \in \mathbb{Z}_n{}^*$. The values $\mathcal{E}, P, Q, n$, and h are the public key and m is the private key.

For $K = (\mathcal{E}, P, m, Q, n, h)$, for a (secret) random number $k \in \mathbb{Z}_n{}^*$, and for a plaintext $x \in \mathbb{Z}_p$, define

$$e_K(x, k) = (\text{POINT-COMPRESS}(kP), x + h(kQ) \bmod p).$$

For a ciphertext $y = (y_1, y_2)$, where $y_1 \in \mathbb{Z}_p \times \mathbb{Z}_2$ and $y_2 \in \mathbb{Z}_p$, define

$$d_K(y) = y_2 - h(R) \bmod p,$$

where

$$R = m\,\text{POINT-DECOMPRESS}(y_1).$$

Hệ mật EEC

```python
def ecc_encrypt(M, a, b, p, P, B):
    # M = double_and_add(x, P, p, a)
    print("M:", M)
    k = randrange(1, 100, 1)

    M1 = double_and_add(k, P, p, a)
    M2 = ecc_add(M[0], M[1], double_and_add(k, B, p, a)[
                0], double_and_add(k, B, p, a)[0], p, a)
    return M1, M2
```

Mã hóa hệ mật EEC

```python
def ecc_decrypt(M1, M2, s, p, a):
    return ecc_add(M2[0], M2[1], double_and_add(s, M1, p, a)[0], -double_and_add(s, M1, p, a)[0], p, a)
```

Giải mã hệ mật EEC

3.2.6. Cài đặt chữ ký hệ mật ElGama Elliptic Curve

Cryptosystem 8.5: *Elliptic Curve Digital Signature Algorithm*

Let p be a large prime and let $\mathcal{E}$ be an elliptic curve defined over $\mathbb{Z}_p$. Let A be a point on $\mathcal{E}$ having prime order q, such that the **Discrete Logarithm** problem in $\langle A \rangle$ is infeasible. Let $\mathcal{P} = \{0,1\}^*$, $\mathcal{A} = {\mathbb{Z}_q}^* \times {\mathbb{Z}_q}^*$, and define

$$\mathcal{K} = \{(p, q, \mathcal{E}, A, m, B) : B = mA\},$$

where $0 \leq m \leq q-1$. The values p, q, $\mathcal{E}$, A, and B are the public key, and m is the private key.

For $K = (p, q, \mathcal{E}, A, m, B)$, and for a (secret) random number k, $1 \leq k \leq q-1$, define

$$\mathbf{sig}_K(x,k) = (r,s),$$

where

$$\begin{aligned} kA &= (u,v) \\ r &= u \bmod q, \quad \text{and} \\ s &= k^{-1}(\text{SHA3-224}(x) + mr) \bmod q. \end{aligned}$$

(If either $r = 0$ or $s = 0$, a new random value of k should be chosen.)

For $x \in \{0,1\}^*$ and $r, s \in {\mathbb{Z}_q}^*$, verification is done by performing the following computations:

$$\begin{aligned} &w = s^{-1} \bmod q \\ &i = w \times \text{SHA3-224}(x) \bmod q \\ &j = wr \bmod q \\ &(u,v) = iA + jB \\ &\mathbf{ver}_K(x,(r,s)) = true \Leftrightarrow u \bmod q = r. \end{aligned}$$

Chữ ký hệ mật ElGama Elliptic Curve

```python
def ecc_signature(d, a, n, p, M, P):
    r1 = 0
    s1 = 0
    while True:
        k = randrange(1, 10000, 1)
        (x1, y1) = double_and_add(k, P, p, a)
        r1 = x1 % n
        if r1 == 0:
            continue
        h = M[0]
        s1 = s1 = ((h + d*r1)*modinv(k, n)) % n
        if s1 == 0:
            continue
        break
    return (r1, s1)
```

Cài đặt chữ ký hệ mật ElGama Elliptic Curve

```python
def ecc_verify(s, r, a, n, M, P, Q):
    w = modinv(s, n)
    h = M[0]
    u1 = (h*w) % n
    u2 = (r*w) % n
    uP = double_and_add(u1, P, n, a)
    uQ = double_and_add(u2, Q, n, a)
    T = ecc_add(uP[0], uP[1], uQ[0], uQ[1], p, a)
    return T[0] == r
```

Cài đặt xác minh hệ mật ElGama Elliptic Curve

3.3. Kết quả:

```
✘ chuis@chuis-HP-EliteBook-8570w ~/Desktop/Test Project/ATTT  master ± python3 ecc.py
Enter Alice secret key:1234
Public key of Alice is (a, b, p, P, B): ( 53 , 7 , 11145971195062230262655792590362754691263974084 11 , (2, 1
1) , (14133338916844917625251682195860756072480957952 9, 17041868777015650485135807342311288784324986523 9) )
Private key of Alice is (s): 1234
--------
Enter Bob secret key:23423
Public key of Bob is (a, b, p, P, B): ( 53 , 7 , 11145971195062230262655792590362754691263974084 11 , (2, 11)
 , (14133338916844917625251682195860756072480957952 9, 17041868777015650485135807342311288784324986523 9) )
Private key of Bob is (d): 23423
--------
Alice want to send message to Bob
Enter point to encrypt:6,5
M: (6, 5)
Alice send to Bob coded message: (59236719363703149482554783882962222320959783331 3, 10455054834279986603988 8615931001373116926627 74) , (10059087201803848100954502542654118157778291080 63, 64104699994166106548141767900
60115999302225559941) )
--------
Bob received message
Verify: True
Message: (6, 5)
```