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. A chuẩn bị và gửi B khóa công khai

- **<u>Bước 1</u>**: Chọn 2 số nguyên tố p, q có độ lớn 512 bit bằng hàm generate_prime_number(512) thu được
    - p=9607148119662215966953860291134935552021529970948227821414276310259699224328420089251860001734469581369286662661929489456209928679379851155917574132015383
    - q=7942814997448628785890615271902035560839675652138142370694028632742919644785493862232366756328919273314953243481427167839097110118579835974997476662633073
    - Từ đó, tính được n = pq và Φ(n) = (p-1)(q-1)
- **<u>Bước 2</u>**: Chọn giá trị b (1<b<Φ(n)) sao cho gcd(b,Φ(n)) = 1. Để đơn giản, chọn b là số nguyên tố bằng hàm generate_prime_number(10) thu được: b=673
- **<u>Bước 3</u>**: Tính được a = b^{-1} mod Φ(n) bằng hàm modinv(b, Φ(n)). Khi đó ta thu được, khóa công khai (n, b) và khóa bí mật (p, q, a)

1.1.2. B mã hóa dựa vào khóa công khai rồi gửi cho A bản mã

- **<u>Bước 4</u>**: Yêu cầu B nhập nội dung cần mã hóa và thực hiện mã hóa bằng hàm rsa_encrypt(x, n, b) rồi gửi kết quả này cho A

1.1.3. A giải mã thu được bản rõ

- **<u>Bước 5</u>**: 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, $n = (a_{k-1}a_{k-2}\dots a_1a_0)_2$,
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 đặt 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

1.3. Kết quả:

```
chuis@chuis-HP-EliteBook-8570w  ~/Desktop/Test Project/ATTT   master ±+ >M<  python3 rsa.py
Enter number to encrypt:123456
Public key is (n,b): ( 763078001675634427533693453219275568907551760087318273499920055926495395829533696
65224361903500030232431517492753739492501602385505581372075034580084159357613696998672389142059677973425
42622310953592292140723017513590485813061020742335839459459656674681237431093603911294466122140208208795
8499975968920561959 , 673 )
Private key is (p,q,a): ( 960714811966221596695386029113493555202152997094822782141427631025969922432842
00892518600017344695813692866626619294894562099286793798511559175741320153 83 , 794281499744862878589061 5
271902035560839675652138142370694028632742919644785493862232366756328919273314953243481427167839097110 11
857983597499747666263 3073 , 238107548814090980359696322698436655974124620532447009561639244791328429605 0
550910801948885547549234592959684023519360092917756457083519782430499527143159199611309825843138445050 89
244897111042379633922219283476880156047394613159429969917550924904691012943758022530135346513326032931 49
57940117013787935578 2817 )
RSA encryption is 57719136236228973499087058951186702668110920323472444612943958501690367861652790064677
72925020083735951276423424897478593363769064259231923462537693874813168814924934683065763182959345965491
77331643629841307555017559833332714207134224690321817685348896866616123573255664630424276145196573980314
15811900776315
RSA decryption is 123456
```

2. Hệ mật ElGama với p - 256 bit

2.1. Quy trình

2.1.1. A chuẩn bị và gửi B khóa công khai

- **<u>Bước 1</u>**: Chọn số nguyên tố p có độ lớn 256 bit bằng hàm generate_prime_number(256) thu được p=10177266595468520050709893144338934308254327994719058886840675548864 3105839567
- **<u>Bước 2</u>**: Tìm phần tử nguyên thủy α bằng hàm primitive(p) thu được α = 5
- **<u>Bước 3</u>**: A nhập khóa bí mật a. Từ đó tính được $\beta = \alpha^a$ (mod p). Khi đó ta thu được khóa công khai (p, α, β) và khóa mật (a)

2.1.2. B mã hóa dựa vào khóa công khai rồi gửi cho A bản mã

- **<u>Bước 4</u>:** Sinh tự động số k bất kì (bí mật) bằng hàm k = randrange(0, 10000000, 1)
- **<u>Bước 5</u>**: Yêu cầu B nhập nội dung cần mã hóa và thực hiện mã hóa bằng hàm elgama_encrypt(x, α, β, p) thu được (y1, y2) rồi gửi cho A

2.1.3. A giải mã thu được bản rõ

- **<u>Bước 5</u>**: 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

1.3. Kết quả:

```
 chuis@chuis-HP-EliteBook-8570w  ~/Desktop/Test Project/ATTT   master ±+ >M<  python3 elgama.py
Choose secret key a:468
Enter number to encrypt:456765
Public key is (p, alpha, beta): 101772665954685200507098931443389343082543279947190588868406755488643105839567 , 5 , 90684342030756202639281880509272135734168360960375818284814994643176378379993
Private key is (a): 468
ElGama encryption is ( 99154719833236669659328613090494245620661858581222526067978184740562344290998 , 60952003741250979115983113473761343423858748507389759459193220448468140456874 )
ElGama decryption is  456765
```

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

3.1. Quy trình

3.1.1. A chuẩn bị và gửi B khóa công khai

- **<u>Bước 1</u>**: 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.
- **<u>Bước 2</u>**: Chọn điểm đầu tiên P của đường Elliptic làm điểm sinh qua hàm findFirstPoint(a, b, p)
- **<u>Bước 3</u>**: A 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)

3.1.2. B mã hóa dựa vào khóa công khai rồi gửi cho A bản mã

- **<u>Bước 4</u>:** Yêu cầu B nhập điểm M cần mã hóa và thực hiện mã hóa bằng hàm ecc_encrypt(M, a, b, p, P, B) thu được (M1, M2) rồi gửi cho A

3.1.3. A giải mã thu được bản rõ

- **<u>Bước 5</u>**: Giải 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 = 111459711950622302626557925903627546912639740841
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≠-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

```python
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

```python
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

3.3. Kết quả:

```
 chuis@chuis-HP-EliteBook-8570w  ~/Desktop/Test Project/ATTT  master ±+ >M<  python3 ecc.py
Enter secret key:34562
Enter point to encrypt:6,9
Public key is (a, b, p, P, B): ( 53 , 7 , 111459711950622302626557925903627546912639740841 1 , (2, 11) , (333
241368637303187620526293020692096070339555 44, 44082297799907420344348374313306753254263204326 3) )
Private key is (s): 34562
M: (6, 9)
EEC encryption is( (3857293446284407432431947071895067761299728455 80, 7139172825614349731218930371356282090 7
782713589) , (610512616511797313923920394745234644408279295 5, 10239555311275599582052106427225540907369150 98
030) )
EEC decryption is (6, 9)
```