CNS LAB 8

SAHIL BONDRE: U18CO021

Write a program to implement the Digital Signature Standard (DSS) algorithm.

```python
from random import randrange



def recursive_read(allowed_input, message=""):

    # Recursively reads user input until input is not in allowd_input

    while True:

        user_input = input(message)

        if user_input in allowed_input:

            return user_input



def isPrime(n):

    if (n <= 1):

        return False

    for i in range(2, n):

        if (n % i == 0):

            return False

    return True



def squareAndMultiply(a, e, m):

    result = 1
```

```python
    while (e > 0):
        if (e % 2 == 1):
            result = (result * a) % m
        e = e >> 1
        a = (a * a) % m
    return result



def multiplicativeInverse(a, m):
    for i in range(1, m):
        remainder = ((i * m) + 1) % a
        if remainder == 0:
            return ((i * m) + 1) // a

    return 0



def generationSig(hash, privateKey):
    print(privateKey)
    p = privateKey[0]
    q = privateKey[1]
    g = privateKey[2]
    x = privateKey[3]
    k = randrange(q)
    r = squareAndMultiply(g, k, p)
    r = r % q
```

```python
    if (r < 0):
        r = q - (r * -1)
    kInv = multiplicativeInverse(k, q)
    hexNum = hash
    s1 = x * r
    s2 = hexNum + s1
    s = (kInv * s2) % q
    if (s < 0):
        s = q - (s * -1)
    return (r, s)


def verificationSig(hash, r, s, publicKey):
    p = publicKey[0]
    q = publicKey[1]
    g = publicKey[2]
    y = publicKey[3]
    w = multiplicativeInverse(s, q)
    hexNum = hash
    u1 = (hexNum * w) % q
    u2 = (r * w) % q
    if (u1 < 0):
        u1 = q - (u1 * -1)
    if (u2 < 0):
        u2 = q - (u2 * -1)
    # (a ^ b * p ^ q) mod m => ((a ^ b) mod m * (p ^ q) mod m) mod m
```

```python
    m1 = squareAndMultiply(g, u1, p)

    m2 = squareAndMultiply(y, u2, p)

    m = (m1 * m2) % p

    if (m < 0):

        m = p - (m * -1)

    v = m % q

    if (v < 0):

        m = q - (v * -1)

    if (v == r):

        print("Signature verified")

    else:

        print("Signature unverified")



def generateKey(q):

    res = []

    p = 2

    while (not(isPrime(p) and (p-1) % q == 0)):

        p += 1

    h = p - 2

    exp = (p - 1) // q

    while (squareAndMultiply(h, exp, p) < 1):

        h -= 1

    g = squareAndMultiply(h, exp, p)

    x = randrange(g)

    y = squareAndMultiply(g, x, p)
```

```python
    p1 = [p, q, g, x]

    p2 = [p, q, g, y]

    res.append(p1)

    res.append(p2)

    return res



flag = False

choice = -1

publicKey = []

privateKey = []

signature = []

print("You must generate the keys before encrypting or decrypting a number ")

generate = recursive_read(["y", "n"], "Do your want to genereate key (y/n): ")

while (generate != "y"):

    print("You must generate the keys before encrypting or decrypting a number
")

    generate = recursive_read(

        ["y", "n"], "Do your want to genereate key (y/n): ")


q = int(input("Enter prime number (q): "))

if (isPrime(q)):

    res = generateKey(q)

    publicKey = res[0]

    privateKey = res[1]

    print("Private Key: ")

    l = 4
```

```python
    print("{", end="")
    for i in range(4):
        if (i < l - 1):
            print(f"{publicKey[i]} ,", end="")
        else:
            print(publicKey[i], end="")
    print("}")

    print("Public Key: ")
    l = 4
    print("{", end="")
    for i in range(4):
        if (i < l - 1):
            print(f"{privateKey[i]} ,", end="")
        else:
            print(privateKey[i], end="")
    print("}")
else:
    print("You must enter a prime number")
    flag = True

while (not flag):
    print("1. Signature Generation")
    print("2. Signature Verification")
    choice = int(input("Enter choice: "))
    if (choice == 1):
```

```python
        hash = int(input("Enter hash of M: "))

        signature = generationSig(hash, publicKey)

        print(f"Signature (r, s): ({signature[0]}, {signature[1]})")

    elif (choice == 2):

        hash = int(input("Enter hash of M: "))

        r = int(input("Enter signature of r: "))

        s = int(input("Enter hash of s: "))

        verificationSig(hash, r, s, privateKey)


    flag = recursive_read(

        ["y", "n"], "Do you want to continue? (y/n): ") == "n"
```

```
PS F:\code\github.com\godcrampy\college-notes\cns\lab-09> python .\main.py
You must generate the keys before encrypting or decrypting a number
Do your want to genereate key (y/n): y
Enter prime number (q): 97
Private Key:
{389 ,97 ,16 ,0}
Public Key:
{389 ,97 ,16 ,1}
1. Signature Generation
2. Signature Verification
Enter choice: 1
Enter hash of M: 56
[389, 97, 16, 0]
Signature (r, s): (7, 48)
Do you want to continue? (y/n): y
1. Signature Generation
2. Signature Verification
Enter choice: 2
Enter hash of M: 56
Enter signature of r: 7
Enter hash of s: 48
Signature verified
Do you want to continue? (y/n): y
1. Signature Generation
2. Signature Verification
Enter choice: 56
Do you want to continue? (y/n): y
1. Signature Generation
2. Signature Verification
Enter choice: 2
Enter hash of M: 56
Enter signature of r: 7
Enter hash of s: 8
Signature unverified
Do you want to continue? (y/n): n
PS F:\code\github.com\godcrampy\college-notes\cns\lab-09>
```