Emerging Blockchain Models for Digital Currencies

Name: Shaik Aarif

Regd No: 2200033144

Exp 3: Implementation of the Custom Asymmetric Key Encryption Algorithm.

Description:

Key Generation

- The key_generation function generates a public-private key pair using predefined values for prime numbers p and q.
- n is the product of p and q, and g is a primitive root modulo n.
- The private key is a (a secret integer), and the public key consists of g, A (where A = g^a mod n), and n.
- The function returns both the private key (a, n) and public key (g, A, n).

Encryption

- The encrypt function encrypts a message m using the public key.
- A random integer k is selected, and then two values are computed:
  - c1 = g^k mod n (first part of the ciphertext)
  - c2 = (m * A^k) mod n (second part of the ciphertext)
- It returns the encrypted message as a tuple (c1, c2).

Decryption

- The decrypt function decrypts the ciphertext (c1, c2) using the private key (a, n).
- It first calculates S = c1^a mod n, which is the shared secret.
- Then, the modular inverse of S (denoted S^-1 mod n) is computed.
- Finally, the original message m is recovered as m = (c2 * S^-1) mod n.

Main Program

1. **Key Generation:** It calls key_generation to obtain the public and private keys.
2. **Message Conversion:** The original message, "BlockChain", is converted into a numeric format by converting each character to its ASCII value (minus 'A' to shift into a suitable range).

3. **Encryption:** Each numeric value is encrypted using the public key, and the encrypted pairs (c1, c2) are stored in a list res.
4. **Decryption:** The ciphertext (c1, c2) pairs are decrypted using the private key, and the numeric results are stored in resdesc.
5. **Message Reconstruction:** Finally, the decrypted numeric values are converted back to characters (by adding the ASCII value of 'A') and printed as the decrypted message.

```python
from sympy import mod_inverse

def key_generation():

    p = 7
    q = 11
    n = p * q
    g = 5
    a = 3
    A = pow(g, a, n)

    private_key = (a,n)
    public_key = (g, A, n)

    return private_key, public_key

def encrypt(public_key, m):
    g, A, n = public_key
    k = 6

    c1 = pow(g, k, n)
    c2 = (m * pow(A, k, n)) % n

    return c1, c2

def decrypt(private_key, c1, c2):

    a,n = private_key

    S = pow(c1, a, n)
    S_inverse = mod_inverse(S, n)

    m = (c2 * S_inverse) % n

    return m

private_key, public_key = key_generation()

print("Private Key:", private_key)
print("Public Key (g, A, n):", public_key)

orginal_message = "BlockChain"
marr= []
for i in orginal_message:
    marr.append(ord(i)-ord('A'))
# m = 13
print("Original Message in number format:", marr)

res= []
for m in marr:
    c1, c2 = encrypt(public_key, m)
    res.append((c1, c2))

print("Cipher Text: ", res)
```

```python

resdesc = []
for c1, c2 in res:
    decrypted_message = decrypt(private_key, c1, c2)
    resdesc.append(decrypted_message)

print("Decrypted Message:", resdesc)

for i in resdesc:
    i= i+ord('A')
    print(chr(i), end='')
```

Screenshot: