# CNS ASSIGNMENT 3 - IMPLEMENTATION OF RSA

## MIS - 112003151

## NAME - ADWAIT VIPRA

RSA (Rivest–Shamir–Adleman) is a widely used public-key cryptosystem that provides secure data transmission and digital signatures. Here's a brief overview of the RSA algorithm:

1. **Key Generation:**
    - **Key Pair:** RSA uses a pair of keys: a public key for encryption and a private key for decryption.
    - **Prime Numbers:** Select two large prime numbers, ( p ) and ( q ).
    - **Modulus:** Compute ( n = pq ), where ( n ) is used as the modulus for both the public and private keys.
2. **Public Key:**
    - **Exponent Selection:** Choose a public exponent ( e ) that is relatively prime to ( (p-1)(q-1) ), typically selecting ( e ) as a small prime number (often 65537).
    - **Public Key ( (e, n) ):** The public key consists of ( e ) and ( n ).
3. **Private Key:**
    - **Private Exponent ( d ):** Compute the private exponent ( d ) such that ( ed \equiv 1 \pmod{(p-1)(q-1)} ).
    - **Private Key ( (d, n) ):** The private key consists of ( d ) and ( n ).
4. **Encryption:**
    - **Message Representation:** Represent the plaintext message as an integer ( m ) where ( 0 \leq m < n ).
    - **Ciphertext Calculation:** Compute the ciphertext ( c \equiv m^e \pmod{n} ).
5. **Decryption:**
    - **Ciphertext Decryption:** Compute the original message ( m \equiv c^d \pmod{n} ).

RSA is widely used for secure communication and digital signatures. Its security is based on the difficulty of factoring the product of two large prime numbers. The security strength of RSA relies on the key length, with longer keys providing higher security but requiring more computational resources.

In [1]:

```
import random
from math import sqrt
from random import randint as rand

def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

def mod_inverse(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return -1

def isprime(n):
    if n < 2:
        return False
    elif n == 2:
        return True
    else:
        for i in range(2, int(sqrt(n)) + 1, 2):
            if n % i == 0:
                return False
    return True

#initialize two random numbers p,q
p = rand(1, 1000)
q = rand(1, 1000)


def generate_keypair(p, q, keysize):
    primes = [2]

    n_min = 1 << (keysize - 1)
    n_max = (1 << keysize) - 1

    start = 1 << (keysize // 2 - 1)
    stop = 1 << (keysize // 2 + 1)

    if start >= stop:
        return []

    for i in range(3, stop + 1, 2):
        for p in primes:
            if i % p == 0:
                break
        else:
            primes.append(i)

    while (primes and primes[0] < start):
        del primes[0]

    #choosing p and q from the generated prime numbers.
    while primes:
        p = random.choice(primes)
        primes.remove(p)
        q_values = [q for q in primes if n_min <= p * q <= n_max]
        if q_values:
```

```
            q = random.choice(q_values)
            break
    print ("p =", p, "\nq =", q)
    n = p * q
    phi = (p - 1) * (q - 1)

    #generate public key 1<e<phi(n)
    e = random.randrange(1, phi)
    g = gcd(e, phi)

    while True:
        #as long as gcd(1,phi(n)) is not 1, keep generating e
        e = random.randrange(1, phi)
        g = gcd(e, phi)

        #generate private key
        d = mod_inverse(e, phi)
        if g == 1 and e != d:
            break

    #public key (e,n)
    #private key (d,n)

    return ((e, n), (d, n))

def encrypt(msg_plaintext, package):
    e, n = package
    msg_ciphertext = [pow(ord(c), e, n) for c in msg_plaintext]

    return msg_ciphertext

def decrypt(msg_ciphertext, package):
    d, n = package
    msg_plaintext = [chr(pow(c, d, n)) for c in msg_ciphertext]

    return (''.join(msg_plaintext))

if __name__ == "__main__":
    bit_length = int(input("Enter Key Length: "))
    print("\nExecuting RSA...")
    print("Generating Public/Private Keypair...\n")

    public, private = generate_keypair(p, q, 2**bit_length)  # 8 is
the keysize (bit-length) value.

    print("\nPublic Key: ", public)
    print("Private Key: ", private)

    msg = input("\nEnter Message: ")

    encrypted_msg = encrypt(msg, public)

    print("\nEncrypted Message: ", end = '')
    print(''.join(map(lambda x: str(x), encrypted_msg)))
    print("\nDecrypted Message: ", end = '')
    print(decrypt(encrypted_msg, private))
```

```
Enter Key Length: 4

Executing RSA...
Generating Public/Private Keypair...

p = 241
q = 193

Public Key:  (8017, 46513)
Private Key:  (31153, 46513)

Enter Message: Being Confidential

Encrypted Message: 18277305861178644473456971521514987446964447315789117862653430586444734500117862615132532

Decrypted Message: Being Confidential
```