# 1. Encryption and Decryption Modules

```python
def Encrypt(m, n, e):
    c = PowMod(ConvertToInt(m),e, n)
    return c

def Decrypt(c, p, q, e):
    phi = (p-1) * (q-1)
    d = InvertModulo(e , phi)
    m = ConvertToStr(PowMod( c , d , p*q))
    return m
```

[538] ✓ 0.4s Python

```python
p = 1000000007
q = 1000000009
exponent = 23917
modulo = p * q
ciphertext = Encrypt("attack", modulo, exponent)
message = Decrypt(ciphertext, p, q, exponent)
print(message)
```

[539] ✓ 0.4s Python

```
... attack
```

# 2. Simple Attack

```python
def DecipherSimple(c, n, e, potential_messages):
    decipheredtext = ''
    for i in range(len(potential_messages)):
        c_potential_messages = Encrypt(potential_messages[i], n, e)
        if c == c_potential_messages:
            decipheredtext = potential_messages[i]
            break
    return decipheredtext
```

[540] ✓ 0.4s Python

```python
modulo = 101
exponent = 12
ciphertext = Encrypt("attack", modulo, exponent)
print(DecipherSimple(ciphertext, modulo, exponent, ["attack", "don't attack", "wait"]))
```

[541] ✓ 0.3s Python

```
... attack
```

# 3. Small Prime Attack

```python
def DecipherSmallPrime(c, n, e):
    decipheredtext = ''
    if n % 2 == 0:
      decipheredtext = Decrypt(c, 2, n // 2, e)
    else :
      for i in range(3, 10**6, 2):
        if n % i == 0:
          decipheredtext = Decrypt(c, i, n // i, e)
          break
    return decipheredtext
```

[542] ✓ 0.4s Python

```python
modulo = 101 * 18298970732541109011012304219376080251334480295537316123696052970419466495220522723330315111017831737980079504337868198011077274303193766040393
exponent = 239
ciphertext = Encrypt("attack", modulo, exponent)
print(DecipherSmallPrime(ciphertext, modulo, exponent))
```

[543] ✓ 0.9s Python

```
... attack
```

## 4. Small Difference Attack

```python
def DecipherSmallDiff(c, n, e):
    decipheredtext = ''
    sqrtN = int(round(np.sqrt(float(n))))
    for i in range(sqrtN - 5000 , sqrtN ):
        if n % i == 0:
            decipheredtext = Decrypt(c, i, n // i, e)
            break
    return decipheredtext
```

[544] ✓ 0.3s Python

```python
p = 1000000007
q = 1000000009
n = p * q
e = 239
ciphertext = Encrypt("attack", n, e)
message = DecipherSmallDiff(ciphertext, n, e)
print(message)
```

[545] ✓ 0.4s Python

```
... attack
```

## 5. Common Divisor Attack

```python
def DecipherCommonDivisor(c1, n1, e1, c2, n2, e2):
    p = GCD(n1 , n2)
    q1 = n1 // p
    q2 = n2 // p
    first_decipheredtext  = Decrypt(c1 , p , q1 , e1)
    second_decipheredtext = Decrypt(c2 , p , q2 , e2)
    return first_decipheredtext, second_decipheredtext
```

[546] ✓ 0.6s Python

```python
p = 101
q1 = 1829897073254110901101230421937608025133448029553731612369605297041946649522052272333031511101783173798007950433786819801107727430319376604039300964885284
q2 = 1000000007
first_modulo = p * q1
second_modulo = p * q2
first_exponent = 239
second_exponent = 17
first_ciphertext = Encrypt("attack", first_modulo, first_exponent)
second_ciphertext = Encrypt("wait", second_modulo, second_exponent)
print(DecipherCommonDivisor(first_ciphertext, first_modulo, first_exponent, second_ciphertext, second_modulo, second_exponent))
```

[547] ✓ 0.1s Python

```
... ('attack', 'wait')
```

## 6. Hastad's Attack

```python
def DecipherHastad(c1, n1, c2, n2, e):
    broadcastmessage = ''
    n = n1 * n2
    N1 = n // n1
    N2 = n // n2
    x1 = InvertModulo(N1 , n1)
    x2 = InvertModulo(N2 , n2)
    c = (c1 * N1 * x1 + c2 * N2 * x2 )%n
    m = int(round(np.power(c , 1./e)))
    broadcastmessage = (ConvertToStr(m))
    return broadcastmessage
```

[548] ✓ 0.3s Python

```python
p1 = 790383132652258876190399065097
q1 = 662503581792812531719955475509
p2 = 656917682542437675078478868539
q2 = 1263581691331332127259083713503
n1 = p1 * q1
n2 = p2 * q2
e = 2
ciphertext1 = Encrypt("attack", n1, e)
ciphertext2 = Encrypt("attack", n2, e)
message = DecipherHastad(ciphertext1, n1, ciphertext2, n2, e)
print(message)
```

[549] ✓ 0.4s Python

```
... attack
```