🔐 RSA Challenge Writeup

🎯 The Challenge

This is a fun RSA challenge with an interesting twist! Instead of the typical approach of factoring the modulus N, we need to recover the message directly using a clever mathematical insight.

What we are given:

- Public key (e, N)
- Ciphertext of the flag: $ct_1 = m^e \bmod N$
- **The interesting part:** $ct_2 = m^{p+q} \bmod N$

Key Observation: The primes p, q are super large, so we can't just factor N. But the leak $ct_2 = m^{p+q} \bmod N$ gives us a different attack vector!

💡 The Solution Approach

Instead of trying to factor N, my approach was to recover the message directly using the leaked information.

Step 1:

Understanding Euler's Totient Function

The key insight starts with Euler's totient function $\phi(N)$:

$$\begin{aligned}\phi(N) &= (p-1)\cdot(q-1)\\ &= pq - p - q + 1\\ &= N + 1 - (p+q)\end{aligned}$$

What is $\phi(N)$? It's called the Euler quotient of N, equal to the product of its prime factors minus 1.

For prime factors $p_1, p_2, \ldots, p_n$ of N:

$$\phi(N) = \prod_{i=1}^{n}(p_i - 1)$$

Step 2:

Euler's Theorem

We know from Euler's theorem that:

$$a^{\phi(N)} \equiv 1 \pmod{N}$$

This means:

$$a^{\phi(N)+1} \equiv a \pmod{N}$$

Insight: If we could raise the plaintext (flag) to $\phi(N) + 1$, we get back the flag!

Step 3:

The Mathematical Trick

Since $\phi(N) = N + 1 - (p+q)$ and we have $m^{p+q} \bmod N$, we can work backwards.

We need to find a way to compute $m^{\phi(N)+1} \bmod N$ without directly knowing m.

Let's express N in terms of e:

$$\begin{gathered}N = \alpha \cdot e + \beta \text{ where } \beta < e\\ \text{Define } \gamma = e - \beta - 1\\ \text{Thus: } e = \gamma + \beta + 1\end{gathered}$$

Step 4:

The Key Calculation

If we raise ct_1 to $\alpha + 1$ and multiply by the inverse of ct_2:

$$\begin{aligned}m_{to\_\gamma} &= ct_1^{\alpha+1} \cdot ct_2^{-1} \bmod N\\ &= m^{e(\alpha+1)} \cdot m^{-(p+q)} \bmod N\\ &= m^{e\alpha+e-(p+q)} \bmod N\\ &= m^{e\alpha+\gamma+\beta+1-(p+q)} \bmod N\\ &= m^{N+\gamma+1-(p+q)} \bmod N\\ &= m^{\phi(N)+\gamma} \bmod N\\ &= m^{\phi(N)} \cdot m^{\gamma} \bmod N\\ &= 1 \cdot m^{\gamma} \bmod N\\ &= m^{\gamma} \bmod N\end{aligned}$$

Step 5:

Recovering the Message with Bézout's Lemma

Now we have:

- $ct_1 = m^e \bmod N$
- $m_{to\_\gamma} = m^{\gamma} \bmod N$

Bézout's Lemma: For given integers a, b, there exist unique integers x, y such that:

$$ax + by = \gcd(a, b)$$

These can be found using the Extended Euclidean Algorithm.

Since $\gamma < e$ and e is prime, we have $\gcd(e, \gamma) = 1$.

We find x, y such that $ex + \gamma y = 1$, then:

$$m = (ct_1^x \cdot m_{to\_\gamma}^y) \bmod N$$

Why does this work?

$$\begin{aligned}(ct_1^x \cdot m_{to\_\gamma}^y) \bmod N &= (m^{ex} \cdot m^{\gamma y}) \bmod N\\ &= m^{ex+\gamma y} \bmod N\\ &= m^1 \bmod N = m \bmod N\end{aligned}$$

🖥️ Implementation

```python
from Crypto.Util.number import *

# Given values
e = 65537
N = 131726351382102866409332377460720737281988694404402738615146884224301154505969635026272696

ct1 = 8499526321488266762028127474977263983474334713646962923180757984708039537289636737028409

ct2 = 2263178005282615069738169250508811825030372342139636879043114251227029802177975391784856

# Calculate alpha, beta, gamma
alpha = N // e
beta = N % e
gamma = e - beta - 1

print(f"{alpha = }")
print(f"{beta = }")
print(f"{gamma = }")

# Calculate m^gamma
m_to_gamma = (pow(ct1, alpha + 1, N) * pow(ct2, -1, N)) % N

def extended_gcd(a, b):
    """
    Returns (gcd, x, y) such that:
       a * x + b * y = gcd(a, b)
    """
    if a == 0:
        return (b, 0, 1)
    else:
        gcd, x1, y1 = extended_gcd(b % a, a)
        x = y1 - (b // a) * x1
        y = x1
        return (gcd, x, y)

# Use Extended Euclidean Algorithm
gcd, x, y = extended_gcd(e, gamma)

# Recover the flag
flag_to_gcd = (pow(ct1, x, N) * pow(m_to_gamma, y, N)) % N

print("Here is the flag to the power of", gcd, ":", long_to_bytes(flag_to_gcd))
```

🎉 Result

This approach successfully recovers the original plaintext message (the flag) without needing to factor the large modulus N!

Key Takeaway: Sometimes in cryptography challenges, the most elegant solution doesn't follow the "standard" approach. The additional leak $ct_2 = m^{p+q} \bmod N$ opened up a completely different attack vector using properties of Euler's totient function and the Extended Euclidean Algorithm.