Sum-O-Primes

Challenge Information

Platform	PicoCTF
Category	Cryptography
Difficulty	Hard
Author	Joshua Inscoe
Flag	picoCTF{pl33z_n0_g1v3_c0ngru3nc3_0f_5qu4r35_3921def5}

Challenge Description

We have so much faith in RSA we give you not just the product of the primes, but their sum as well!

Hint: I love squares :)

Files Provided

- gen.py - RSA key generation script
- output.txt - Encrypted flag and public values

Vulnerability Analysis

This challenge presents an interesting RSA vulnerability where the attacker is given both the product ($n = p \times q$) and the sum ($x = p + q$) of the two prime factors. This combination of information makes factoring trivial through a quadratic equation approach.

Mathematical Background

Standard RSA Setup:

- • $n = p \times q$ (modulus - product of two primes)
- • $\varphi(n) = (p-1)(q-1)$ (Euler's totient)
- • $e = 65537$ (public exponent)
- • $d \equiv e^{-1} \bmod \varphi(n)$ (private exponent)

This Challenge Twist:

- Given: $n = p \times q$ (standard)
- Given: $x = p + q$ (unusual!)
- Attack: Solve quadratic equation to recover p and q

Solution Approach

Step 1: Derive Quadratic Equation

From the given values $x = p + q$ and $n = p \times q$, we can derive:

$q = x - p$

Substituting into $n = p \times q$:

$n = p(x - p)$

$n = px - p^2$

$p^2 - px + n = 0$

This is a standard quadratic equation: $ap^2 + bp + c = 0$

Step 2: Apply Quadratic Formula

Using the quadratic formula:

$p = (x + \sqrt{(x^2 - 4n)}) / 2$

$q = (x - \sqrt{(x^2 - 4n)}) / 2$

The discriminant $\Delta = x^2 - 4n$ must be a perfect square for integer solutions. The hint "I love squares" confirms that the discriminant is indeed a perfect square!

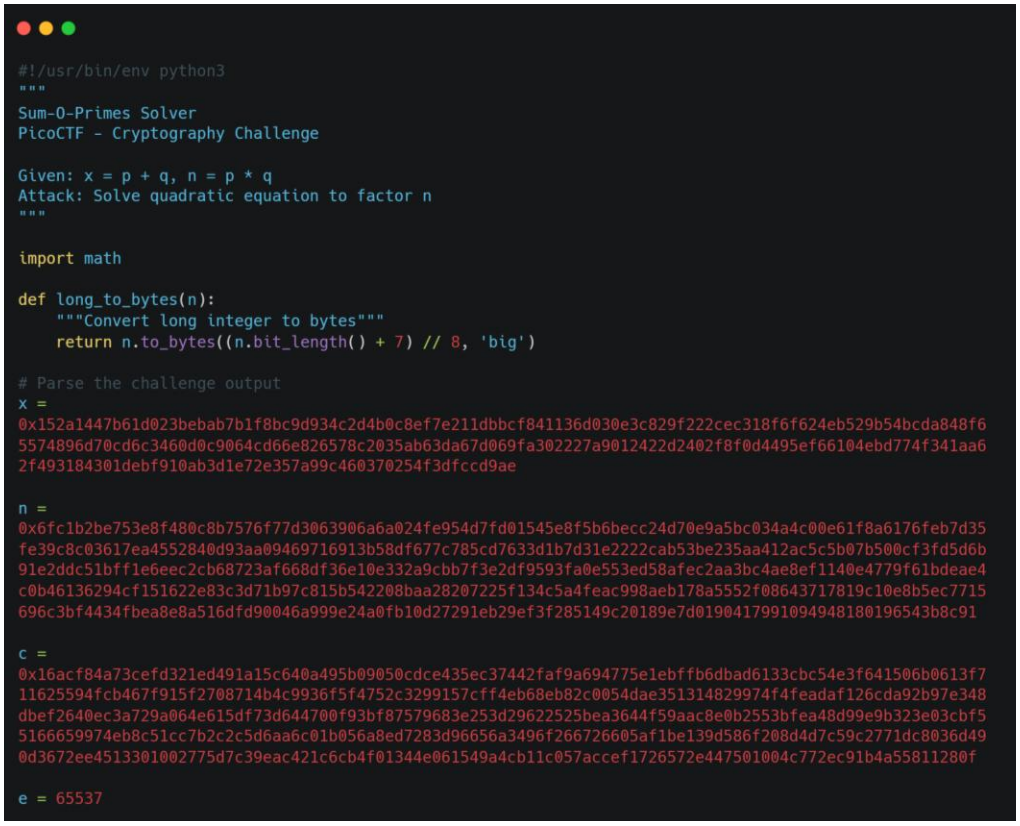
Step 3: Factor and Decrypt

Once we have p and q:

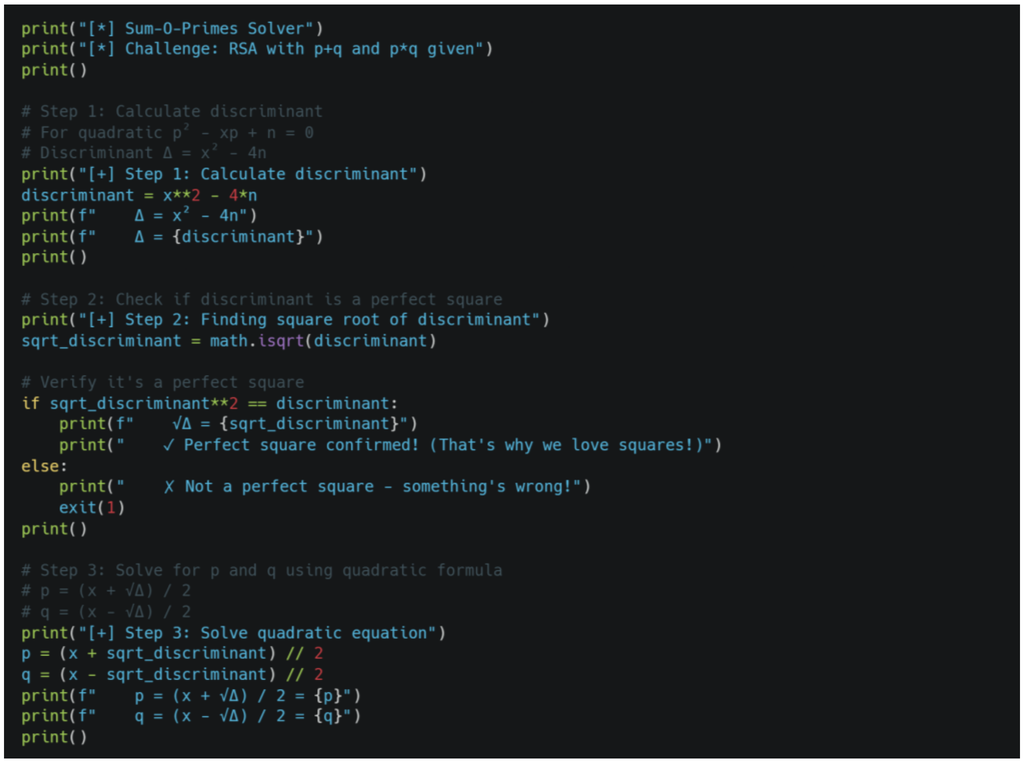
1. 1. Calculate $\varphi(n) = (p-1)(q-1)$
2. 2. Compute private key: $d \equiv e^{-1} \bmod \varphi(n)$
3. 3. Decrypt ciphertext: $m \equiv c$^$d \bmod n$
4. 4. Convert integer to bytes to get the flag

Implementation

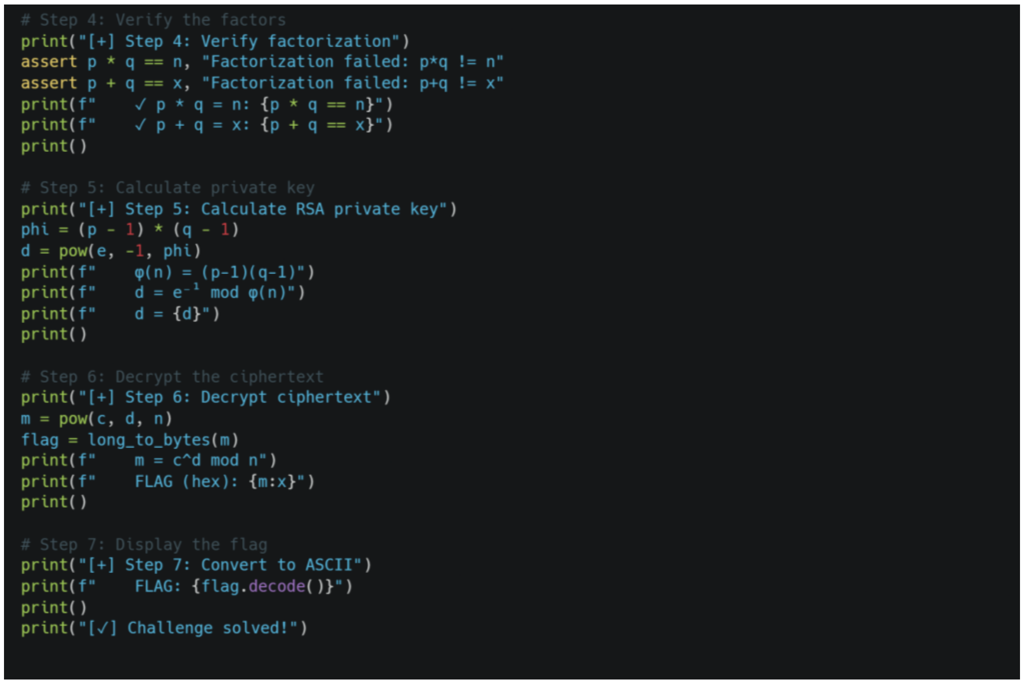
The solve script implements the attack in Python:



```python
#!/usr/bin/env python3
"""
Sum-O-Primes Solver
PicoCTF - Cryptography Challenge

Given: x = p + q, n = p * q
Attack: Solve quadratic equation to factor n
"""

import math

def long_to_bytes(n):
    """Convert long integer to bytes"""
    return n.to_bytes((n.bit_length() + 7) // 8, 'big')

# Parse the challenge output
x =
0x152a1447b61d023bebab7b1f8bc9d934c2d4b0c8ef7e211dbbcf841136d030e3c829f222cec318f6f624eb529b54bcda848f6
5574896d70cd6c3460d0c9064cd66e826578c2035ab63da67d069fa302227a9012422d2402f8f0d4495ef66104ebd774f341aa6
2f493184301debf910ab3d1e72e357a99c460370254f3dfccd9ae

n =
0x6fc1b2be753e8f480c8b7576f77d3063906a6a024fe954d7fd01545e8f5b6becc24d70e9a5bc034a4c00e61f8a6176feb7d35
fe39c8c03617ea4552840d93aa09469716913b58df677c785cd7633d1b7d31e2222cab53be235aa412ac5c5b07b500cf3fd5d6b
91e2ddc51bff1e6eec2cb68723af668df36e10e332a9cbb7f3e2df9593fa0e553ed58afec2aa3bc4ae8ef1140e4779f61bdeae4
c0b46136294cf151622e83c3d71b97c815b542208baa28207225f134c5a4feac998aeb178a5552f08643717819c10e8b5ec7715
696c3bf4434fbea8e8a516dfd90046a999e24a0fb10d27291eb29ef3f285149c20189e7d01904179910949481801965 43b8c91

c =
0x16acf84a73cefd321ed491a15c640a495b09050cdce435ec37442faf9a694775e1ebffb6dbad6133cbc54e3f641506b0613f7
11625594fcb467f915f2708714b4c9936f5f4752c3299157cff4eb68eb82c0054dae351314829974f4feadaf126cda92b97e348
dbef2640ec3a729a064e615df73d644700f93bf87579683e253d29622525bea3644f59aac8e0b2553bfea48d99e9b323e03cbf5
5166659974eb8c51cc7b2c2c5d6aa6c01b056a8ed7283d96656a3496f266726605af1be139d586f208d4d7c59c2771dc8036d49
0d3672ee4513301002775d7c39eac421c6cb4f01344e061549a4cb11c057accef1726572e447501004c772ec91b4a55811280f

e = 65537
```



```python
print("[*] Sum-O-Primes Solver")
print("[*] Challenge: RSA with p+q and p*q given")
print()

# Step 1: Calculate discriminant
# For quadratic p² - xp + n = 0
# Discriminant Δ = x² - 4n
print("[+] Step 1: Calculate discriminant")
discriminant = x**2 - 4*n
print(f"    Δ = x² - 4n")
print(f"    Δ = {discriminant}")
print()

# Step 2: Check if discriminant is a perfect square
print("[+] Step 2: Finding square root of discriminant")
sqrt_discriminant = math.isqrt(discriminant)

# Verify it's a perfect square
if sqrt_discriminant**2 == discriminant:
    print(f"    √Δ = {sqrt_discriminant}")
    print("    ✓ Perfect square confirmed! (That's why we love squares!)")
else:
    print("    ✗ Not a perfect square - something's wrong!")
    exit(1)
print()

# Step 3: Solve for p and q using quadratic formula
# p = (x + √Δ) / 2
# q = (x - √Δ) / 2
print("[+] Step 3: Solve quadratic equation")
p = (x + sqrt_discriminant) // 2
q = (x - sqrt_discriminant) // 2
print(f"    p = (x + √Δ) / 2 = {p}")
print(f"    q = (x - √Δ) / 2 = {q}")
print()
```



```
# Step 4: Verify the factors
print("[+] Step 4: Verify factorization")
assert p * q == n, "Factorization failed: p*q != n"
assert p + q == x, "Factorization failed: p+q != x"
print(f"    ✓ p * q = n: {p * q == n}")
print(f"    ✓ p + q = x: {p + q == x}")
print()

# Step 5: Calculate private key
print("[+] Step 5: Calculate RSA private key")
phi = (p - 1) * (q - 1)
d = pow(e, -1, phi)
print(f"    φ(n) = (p-1)(q-1)")
print(f"    d = e⁻¹ mod φ(n)")
print(f"    d = {d}")
print()

# Step 6: Decrypt the ciphertext
print("[+] Step 6: Decrypt ciphertext")
m = pow(c, d, n)
flag = long_to_bytes(m)
print(f"    m = c^d mod n")
print(f"    FLAG (hex): {m:x}")
print()

# Step 7: Display the flag
print("[+] Step 7: Convert to ASCII")
print(f"    FLAG: {flag.decode()}")
print()
print("[✓] Challenge solved!")
```

Results

Running the solve script successfully factors the RSA modulus and decrypts the flag:

FLAG: picoCTF{pl33z_n0_g1v3_c0ngru3nc3_0f_5qu4r35_3921def5}

Key Takeaways

5. 1. Never leak information about RSA prime factors
6. 2. The combination of p×q and p+q makes factoring trivial
7. 3. Even "harmless" looking information (like a sum) can completely break RSA security
8. 4. This attack works regardless of key size - the 1024-bit primes provide no security here

Real-World Implications

This challenge demonstrates why RSA implementations must carefully protect ALL information related to the prime factors. In real-world systems:

- • Never expose p, q, or φ(n)
- • Never expose relationships like p+q, p-q, or gcd(p-1, q-1)
- • Use proper random number generation for prime selection
- • Implement side-channel attack protections

Tools Used

- Python 3
- Standard library: math module
- Custom function: long_to_bytes() for integer to bytes conversion

Challenge solved by Glenvio Regalito Rahardjo

December 2025