R Stands Alone (Crypto) - NiteCTF2024

NiteCTF (Crypto) - Writeup

- **Solution/Writeup Author**: supasuge
- **Date**: 12/15/2024

This was the only challenge I solved during this event sadly, as I didn't have much time due to mostly family & other obligations.

I was quite disappointed after I realized how easy it really was after spending a lot of time going down various rabbit holes although once all said and done, after working out felt like I learned a lot. Overall a good challenge, lesson learned despite limited time: Don't overcomplicate things!

Challenge Source Code (Preliminary)

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

def gen_keys():
    while True:
        a = getPrime(128)
        b = getPrime(128)
        A = a+b
        B = a-b

        p = ((17*A*A*A) - (15*B*B*B) - (45*A*A*B) + (51*A*B*B)) // 8

        if isPrime(p) :
            return a, b, p
```

```
p, q, r = gen_keys()
e = 65537
n = p*q*r

flag = b"nite{REDACTED}"

ct = pow(bytes_to_long(flag), e, n)
print(f"{r =}")
print(f"{ct =}")

"""OUTPUT :
r =1708972084752253218610090449537295479608652343934340119012357224312990574
ct =583923134770560329725969597854974954817875793223201855918544947864454666
"""
```

Challenge overview

This challenge involves RSA with a twist where $n = p \cdot q \cdot r$ consists of three primes. The value r is generated via a special algebraic construction, and our goal is to factor r using number field techniques to recover p, and q, then decrypt the flag.

Vulnerability Background

Step 1: Understanding the Prime Generation

The server generations primes using this formula below (for a more detailed analysis see section below):

```python
p = ((17*A^3) - (15*B^3) - (45*A^2*B) + (51*A*B^2)) // 8
```

Where $A = a + b$ and $B = a - b$ for randomly generated 128-bit primes a, and b.

Key simplification

- This expression reduces to $r = a^3 + 16b^3$ (proof below). This structure what allows this attack to succeed with such ease.

Here's the algebraic derivation formatted in proper Obsidian markdown with TeX:

Step 2: Algebraic Derivation of $r = a^3 + 16b^3$

Let's expand the given formula with $A = a + b$ and $B = a - b$:

$$17A^3 = 17(a+b)^3 = 17(a^3 + 3a^2b + 3ab^2 + b^3)$$

$$-15B^3 = -15(a-b)^3 = -15(a^3 - 3a^2b + 3ab^2 - b^3)$$

$$-45A^2B = -45(a+b)^2(a-b) = -45(a^3 + a^2b - ab^2 - b^3)$$

$$51AB^2 = 51(a+b)(a-b)^2 = 51(a^3 - a^2b - ab^2 + b^3)$$

Adding these terms and dividing by 8:

$$\frac{17A^3 - 15B^3 - 45A^2B + 51AB^2}{8} = \frac{8a^3 + 128b^3}{8} = a^3 + 16b^3$$

Thus, $r = a^3 + 16b^3$.

Code Analysis

```python
from Crypto.Util.number import *
def gen_keys():
    while True:
        a = getPrime(128)
        b = getPrime(128)
        A = a+b
        B = a-b

        p = ((17*A*A*A) - (15*B*B*B) - (45*A*A*B) + (51*A*B*B)) // 8
```

```
        if isPrime(p):
            return a, b, p
```

- a Random 128-bit prime number
- b: Random 128-bit prime number
- A: The sum of ($a+b$)
- B: Difference of ($a-b$)

$$p = \frac{17A^3 - 15B^3 - 45A^2B + 51AB^2}{8}$$

If p is a prime number, it returns $a,\ b,\ p$

Vulnerability

The key insight here is that the prime r can be factorized in $\mathbb{Q}(a)$, where a is a root of $x^3 - 16$. This means $a^3 = 16$. The ring of integers of this field, denoted as $\mathcal{O}_K$, contains elements of the form $p + qa$ where $p,\ q$ are rational integers.

The Norm Map

A crucial concept here is the norm of an element in this number field. For an element $p + qa$, its norm is: $N(p+qa) = (p+qa)(p+q\omega a)(p+q\omega^2 a)$

where ω is a primitive cube root of unity. When we expand this, we get: $N(p+qa) = p^3 + 16q^3$

And with that, we get the exact form of our prime `r`.

Why this helps with factorization

When we factor `r` in the ring of integers $\mathcal{O}_K$, we're essentially finding the elements whose norm is `r`. Since we know `r` is constructed as $p^3 + 16q^3$, one of these factors must be of the form $p + qa$. The coefficients of this linear factor give us our `p` and `q` directly to properly recover N.

Recovery of the private key

When we factor `r` in the ring of integers $\mathcal{O}_K$, we find elements whose norm is `r`. Since `r` is constructed as $p^3 + 16q^3$, one of these factors must be of the form $p + qa$. The coefficients of this linear factor give us our `p` and `q` directly.

With `p` and `q` recovered, and `r` already known, we can:

1. Calculate the modulus $N = p \cdot q \cdot r$
2. Compute Euler's totient function: $\phi(N) = (p-1)(q-1)(r-1)$
3. Find the private exponent $d \equiv e^{-1} \pmod{\phi(N)}$
4. Decrypt the ciphertext: $pt \equiv ct^d \pmod{N}$

Solution Steps

Setup: The RSA modulus is $n = p \cdot q \cdot r$. One of the primes we are given r, is chosen such that $r = p^3 + 16q^3$ for some unknown p, q.
Number Field Trick: Consider the field $\mathbb{Q}(a)$ with $a^3 = 16....$
In this field, the norm of $p + qa$ is:
$p^3 + 16q^3$ which equals r.
Factorization in the Number Field: Factoring r in this number field (i.e., factoring the ideal (r) in $\mathcal{O}_K$) gives a linear factor that reveals p and q.
Breaking RSA: Once p, q, r are found, compute $\varphi(n) = (p-1)(q-1)$ (phi, Euler's totient function).

- invert e modulo $\varphi(n)$ to find d, and decrypt the ciphertext.

```python
d = pow(e, -1, phi)
# Or using the inverse_mod function
d = inverse_mod(e, phi)
```

SageMath Solution Code

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

r = 17089720847522532186100904495372954796086523439343401190123572243129905
ct = 58392313477056032972596959785497495481787579322320185591854494786445446

# Define our number field Q(a) where a^3 = 16
x = var("x")
K = NumberField(x^3 - 16, "a")
R = K.ring_of_integers()

# Factor r in the ring of integers
factors = R(r).factor()

for factor, exponent in factors:
    # Convert factor to polynomial representation
    f = (factor^exponent).polynomial()

    # We're looking for a linear factor of the form p + qa
    if f.degree() == 1:
        # Extract p (constant term) and q (coefficient of a)
        p = f.constant_coefficient()
        q = f.leading_coefficient()

        # Verify our factorization: r should equal p^3 + 16q^3
        assert p^3 + 16 * q^3 == r

        # Construct RSA parameters
```

```
        n = p * q * r
        phi = (p - 1) * (q - 1) * (r - 1)
        e = 65537

        # Calculate private exponent
        d = inverse_mod(e, phi)

        # Decrypt flag
        m = pow(ct, d, n)
        flag = long_to_bytes(m)
        print(f"Flag: {flag.decode()}")
```

```
sage -python solve.py
Flag: nite{7h3_Latt1c3_kn0ws_Ur_Pr1m3s_very_vvery_v3Ry_w3LLL}
```

LLL is not required here at all .

You can solve easily with LLL, don't get me wrong... But it's unnecessary if you ask me, most crypto challenges are solvable via LLL *if you try hard enough* (lol) so kind of takes the fun out of it is the point I'm getting at...