Arxenix's Blog

Some random math & cs stuff

HOME GITHUB

ASIS CTF Finals 2016 - RACES

11 SEPTEMBER 2016

> *Category:* Crypto
>
> *Points:* 189
>
> *Description:* Find the flag by using the given files.

In this problem, we are given 2 files, `RACES.py` , and `pubkey_enc.txt` .

RACES.py:

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

def gen_prime(nbit):
    while True:
        prime = getPrime(nbit)
        if prime % 3 == 2:
            return prime

def add(a, b, n):
    if a == 0:
        return b
    if b == 0:
        return a
    l = ((b[1] - a[1]) * invert(b[0] - a[0], n)) % n
    x = (l*l - a[0] - b[0]) % n
    y = (l*(a[0] - x) - a[1]) % n
    return (x, y)

def double(a, A, n):
    if a == 0:
        return a
    l = ((3*a[0]*a[0] + A) * invert(2*a[1], n)) % n
    x = (l*l - 2*a[0]) % n
    y = (l*(a[0] - x) - a[1]) % n
    return (x, y)

def multiply(point, exponent, A, n):
    r0 = 0
    r1 = point
    for i in bin(exponent)[2:]:
        if i == '0':
            r1 = add(r0, r1, n)
            r0 = double(r0, A, n)
        else:
            r0 = add(r0, r1, n)
```

```
            r1 = double(r1, A, n)
        return r0

    def gen_keypair(e, nbit):
        p = gen_prime(nbit)
        q = gen_prime(nbit)
        n = p*q
        lcm = (p+1)*(q+1)/GCD(p+1, q+1)
        d = invert(e, lcm)
        pubkey = (n, e)
        privkey = (n, d)
        return pubkey, privkey

    def encrypt(msg, pubkey):
        n, e = pubkey
        if msg < n:
            while True:
                r = getRandomRange(1, n)
                m1, m2 = r - msg, r
                if m1 > 0:
                    break
            c1, c2 = multiply((m1, m2), e, 0, n)
            return (int(c1), int(c2))
        else:
            return 'Error!!!'

    def gen_keypair(e, p, q):
        n = p*q
        lcm = (p+1)*(q+1)/GCD(p+1, q+1)
        d = invert(e, lcm)
        pubkey = (n, e)
        privkey = (n, d)
        return pubkey, privkey
```

pubkey_enc.txt:

This file contained a large list of {(n,e), c} objects. For example:

```
{ (n, e) = (794645807885475483676813754291414647718975579685056902008230510251206
```

Analyzing the encryption algorithm revealed a few things:

Key Generation:

1. Generate 2 n-bit primes, p, and q, such that $p \equiv q \equiv 2 \bmod 3$
2. $N = pq$
3. $d \equiv e^{-1} \bmod \operatorname{lcm}((p+1)(q+1))$
4. $pubkey = (N, e)$
5. $privkey = (N, d)$

So, it seems somewhat like RSA... except instead of $\varphi = (p-1)(q-1)$, it is equal to $\operatorname{lcm}(p+1)(q+1)$

Encryption:

1. Generate a random integer, r such that $1 \leq r < N$
2. $multiply(\langle r - msg, r \rangle, e, 0, N)$

I recognized the `multiply` function as being an implementation of Montgomery Ladder Scalar Multiplication on Elliptic Curves.

So, $\langle r - msg, r \rangle$ is the initial point on the elliptic curve, e is the exponent (and is equal to 65537 in each of the public keys), $A = 0$ for the curve equation, $y^2 = x^3 + Ax + B$, and the field is $GF(N)$.

Overall, the cryptosystem seems to be a combination of ECC and RSA. With some google-fu, I found this paper, which discusses the ECRSA cryptosystem.

Now, I don't fully understand the math behind this cryptosystem, but the key generation and encryption algorithms that the paper describes match the code that we were provided, so we can use the description algorithm that it describes. I hope to spend some time analyzing and understand the math behind this later (and may do another blog post on it)

Anyways, to decrypt, we can simply do $multiply(d, C)$ to recover the message.

Our goal now is to try and figure out d for the messages. This seems impossible at first, because we would need to factor each N, which are each ~300 digits. Considering the fact that there were so many public keys, and the restriction on the primes that $p \equiv q \equiv 2 \bmod 3$, I thought it might be worth a shot to see if some of the N had common factors.

And.... yup! Exactly 2 Ns had a common factor. Here is my final solution code (just appended to the original `RACES.py` file):

```
e = 65537
# The verified p,q recovered pairs based on shared prime
N1 = 14502748278969099026251775095154144622155225556052022870387731343148331674l2
N2 = 11640245266607293620881500713930498721671416304981123478135142632692413124l6
p1 = p2 = gcd(N1,N2)
q1 = N1/p1
q2 = N2/p2
c1 = (84876076421614376067149365902722288787017939432560112310344060253776893355l
c2 = (19643266456009901393757401035383589370137211795890848379404807360451967720l

# Recover the private exponents with the generation function
pubkey, privkey = gen_keypair(e, p1, q1)
d1 = privkey[1]
pubkey, privkey = gen_keypair(e, p2, q2)
d2 = privkey[1]

x,y = multiply(c1, d1, 0, N1)

# Decode the flag
print hex(y-x)[2:].decode('hex')
```

Flag:

`ASIS{58cf105e8993ff852a7ea69c3f6464458a87c69f89ef3dfd749da4e2d3982de34832e38cab1baf8d1cd3ce0`

CTF, CRYPTO

0 Comments

Sort by Oldest

Add a comment...

Facebook Comments Plugin