Unimplemented

January 5, 2021

1 The challenge

This challenge was part of TetCTF 2021. The description of the challenge was:

> "A new public key encryption algorithm is being invented, but the author is not quite sure how to implement the decryption routine correctly. Can you help him?"

The challenge was primarily to analyze the Python implementation of a new public key encryption algorithm whose decryption routine had not yet been implemented and find a way to do so to recover the encrypted flag.

2 The algorithm

The algorithm is based on the well known RSA cryptosystem, but it has two major differences:

- the first one is that instead of using simple integer numbers, it uses Gaussian integers, i.e. complex numbers where both real and imaginary parts are integer values.
- the second is that the modulus N is calculated as P^2Q^2 instead of simply PQ.

The encryption function was as follows:

Algorithm 1 Encryption routine

```
def encrypt(public_key, plaintext):

    n = public_key
    plaintext = pad(plaintext, 2 * ((n.bit_length() - 1) // 8))
    m = Complex(
        int.from_bytes(plaintext[:len(plaintext) // 2], "big"),
        int.from_bytes(plaintext[len(plaintext) // 2:], "big"))
    c = complex_pow(m, 65537, n)
    return (c.re.to_bytes((n.bit_length() + 7) // 8, "big")
          + c.im.to_bytes((n.bit_length() + 7) // 8, "big"))
```

As we can see, the plain text is filled so that its size (in bits) is of the same order of magnitude as the private key. Then the text is divided in half, and each half is converted into a numerical value that is then used to form a complex number where the real part is the first half and the imaginary part is the second one.

Since complex numbers are used, the code includes a function for multiplication and modular exponentiation of such numbers. Nevertheless, the encryption process occurs in the same way as in RSA: given a private key (e, N) and a plain text message M, the cipher text C is calculated as $M^e \mod N$.

In this algorithm, the exponent e is fixed as 65537 and both the modulus N and the constants P and Q are present at the end of the file as comments, so all we need to do is implement the decryption function. Based on similarities with RSA, we can assume that the decryption process should also look at least partially similar.

3 Decryption process

In RSA, decryption would be done as follows:

- Calculate $\phi(N) = (P-1)(Q-1)$
- Calculate $d = invmod(e, \phi(N))$
- Calculate $m = c^e \mod N$

Since N is defined as P^2Q^2 intead of just PQ, we need to figure out the correct way to calculate $\phi(N)$ (also known as Euler's totient function) for this group of Gaussian integers.

To solve this we can start with the definition of ϕ. Given $x = {p_1}^{r_1}{p_2}^{r_2}\dots{p_k}^{r_k}$, where p_i is the i-th prime factor of N and r_i is the number of times N can be divided by such factor, we have that

$$\phi(x) = p_1^{r_1-1}(p_1-1)p_2^{r_2-1}(p_2-1)\dots p_k^{r_k-1}$$

By the Fundamental Theorem of Arithmetic, $\phi(x)$ can also be written as:

$$\phi(x) = x(1-\frac{1}{p_1})(1-\frac{1}{p_2})(1-\frac{1}{p_3})\dots(1-\frac{1}{p_k}) \tag{1}$$

As the totient function is multiplicative, i.e. $\phi(xy) = \phi(x)\phi(y)$, we can write $\phi(N)$ as

$$\phi(N) = \phi(P^2)\phi(Q^2)$$

And so, we have

$$\phi(N) = P^2(1-\frac{1}{P})Q^2(1-\frac{1}{Q}) \tag{2}$$

Easy, right? Wrong... because here comes the real problem: we are not dealing with simple integers, but with Gaussian Integers! Remember that both P and Q are complex numbers with an imaginary part equal to 0.

Luckily, there is a (relatively) simple way to generalize Euler's totient to Gaussian integers. And it consists in using a variantion of (1) which uses a norm function η defined as $\eta(\alpha) = a^2 + b^2$ where a and b are the real and imaginary parts of α, respectively. So our new totient function is:

$$\phi(x^k) = \eta(x)^k(1 - \frac{1}{\eta(x)})$$

By applying this variation to (2), we can calculate $\phi(N)$ using

$$\phi(P^2) = \eta(P)^2(1 - \frac{1}{\eta(P)})$$

$$\phi(Q^2) = \eta(Q)^2(1 - \frac{1}{\eta(Q)})$$

$$\phi(N) = \phi(P^2)\phi(Q^2)$$

$$\phi(N) = \eta(P)^2(1 - \frac{1}{\eta(P)})\eta(Q)^2(1 - \frac{1}{\eta(Q)}) \tag{3}$$

Using (3) we can find

$$\phi(N) = (P^4 - P^2)(Q^4 - Q^2) \tag{4}$$

4 Decryption routine

Having calculated (4)[1], the rest of the decryption routine can be implemented as it would be in RSA. So, our decrypt function would look something like:

[1] Equation (4) can be reduced to $\phi(N) = (P^3 - P)(Q^3 - Q)$ with the same effect.

Algorithm 2 Decryption routine

```
    def decrypt(private_key, ciphertext):
        (p, q) = private_key
        n = (p ** 2) * (q ** 2)
        c = Complex(
            int.from_bytes(ciphertext[:len(ciphertext) // 2], "big"),
            int.from_bytes(ciphertext[len(ciphertext) // 2:], "big")
        )
        e = 65537
        phi = (p ** 4 - p ** 2) * (q ** 4 - q ** 2)
        d = inverse(e, phi)
        m = complex_pow(c, d, n)
        return unpad(m.re.to_bytes((n.bit_length() + 7) // 8, "big")
                     + m.im.to_bytes((n.bit_length() + 7) // 8, "big"))
```

THE END