NTRU Post-Quantum Encryption

Kateryna Makovetska

January 30, 2022

1 Example from Wikipedia

Let's try to re-create `https://en.wikipedia.org/wiki/NTRUEncrypt`. In this example the parameters (N, p, q) will have the values $N = 11$ $p = 3$ $q = 2^5$ The polynomials f and g are randomly chosen, so suppose they are represented by
$f = -1 + x + x^2 - x^4 + x^6 + x^9 - x^{10}$
$g = -1 + x^2 + x^3 + x^5 - x^8 - x^{10}$

1.1 balancedmod(f,q)

This function reduces every coefficient of a $Z[x]$ polynomial f modulo q with additional balancing, so the result coefficients are integers in interval $[-\frac{q}{2}, +\frac{q}{2}]$. More specifically: for an odd q $[-\frac{q-1}{2}, +\frac{q-1}{2}]$, for an even q $[-\frac{q}{2}, +\frac{q}{2} - 1]$. It returns $Z[x]$ reduced polynomial.

```
def balancedmod(f,q):
    g = list(((f[i] + q//2) % q) - q//2 for i in range(N))
    return Zx(g)
```

Example:

$$\text{balancedmod}(17 + 31x + 34x^2 + 33x^3 - 8x^4, 32) = -8x^4 + x^3 + 2x^2 - x - 15$$

1.2 multiply(f,g)

This function performs a multiplication operation specific for NTRU, which works like a traditional polynomial multiplication with additional reduction of the result by $x^N - 1$ (x^n is replaced by 1, $x^n - 1$ by x, $x^n - 2$ by x^2, ...) and returns $Z[x]$ polynomial

```
def convolution(f,g):
    return (f * g) % (x^N-1)
```

1.3 invertmodprime(f,p)

This function calculates an inversion of a polynomial modulo $x^N - 1$ and then modulo p with assumption that p is prime. Also it returns a $Z[x]$ polynomial h such as convolution of $h \quad f = 1$ $(mod$p) and raises an exception if such $Z[x]$ polynomial h doesn't exist.

```
def invertmodprime(f,p):
    Zq.<z> = PolynomialRing(Integers(p))
    ZQphi.<Z> = Zq.quotient(z^N-1)
    a = f % p
    a = a.subs(x=z)
    k = 0
    b = 1*z^0
    c = 0*z^0
    f = a
    g = z^N-1
    assert a.gcd(g) in {i for i in range(p)}

    while True:
        while list(f)[0] == 0:
            f = f / Z
            c = c * Z
            k += 1

        if find_degree(list(f)) == 0:
            b = 1/list(f)[0] * b
            ans = Z^(N-k) * b
            return Zx(ans.lift())

        if find_degree(list(f)) < find_degree(list(g)):
            f, g = g, f
            b, c = c, b

        u = list(f)[0] * (1/list(g)[0])
        f = f - u*g
        b = b - u*c
```

1.4 invertmodpowerof2(a, p)

This function calculates an inversion of a polynomial modulo $x^N - 1$ and then modulo q with assumption that q is a power of 2. Returns a polynomial $f_q \in Z[x]$ such as $f \cdot f_q = 1(\text{mod} q)$. An exception is thrown if such polynomial $Z[x]$ does not exist.

```
def invertmodpowerof2(a, p):
```

```
    r = int(math.log(p, 2))
    p = 2
    q = p
    b = invertmodprime(a, p)
    while q < p^r:
        q = q^2
        b = b * (2 - a*b) % q % (x^N-1)

    b = b % p^r % (x^N - 1)
    return b
```

Example:

$$f = -x^{10} + x^9 + x^6 - x^4 + x^2 + x - 1, p = 3, N = 11, q = 2^5$$

$f_q = \text{invertmodpowerof2}(\text{f}, \text{q}) = 30x^{10}+18x^9+20x^8+22x^7+16x^6+15x^5+4x^4+16x^3+6x^2+9x+5$

Note that this is exactly the inverse mentioned in Wikipedia - NTRU.

1.5 generate_keys(poly1 = None, poly2 = None)

In this section we generate a public and secret key. Function generate_keys(poly1 = None, poly2 = None) generates a public and private key pair, based on provided parameters. Returns $Z[x]$ public key and a secret key as a tuple of $Z[x]$ f (private key) and $Z[x]$ F_p

```
def generate_keys(poly1 = None, poly2= None):
    if validate_params():
        while True:
            try:
                if poly1 is None or poly2 is None:
                    f = generate_polynomial(d+1, d)
                    g = generate_polynomial(d, d)
                else:
                    f = poly1
                    g = poly2
                f_q = invertmodpowerof2(f,q)
                f_p = invertmodprime(f,p)
                break
            except:
                pass
        public_key = balancedmod(p * convolution(f_q,g),q)
        secret_key = f,f_p
        return public_key,secret_key

    else:
        print("Provided params are not correct.")
```

Example: In this example the parameters (N, p, q) will have the values $N = 11$, $p = 3$ and $q = 32$ and therefore the polynomials f and g are of degree at most 10. The system parameters (N, p, q) are known to everybody. The polynomials are randomly chosen, so suppose they are represented by
$f(x) = -1 + x + x^2 - x^4 + x^6 + x^9 - x^{10}$, $g(x) = -1 + x^2 + x^3 + x^5 - x^8 - x^{10}$

$public\_key = \text{get\_keys}(\text{f}, \text{g}) = -16x^{10} - 13x^9 + 12x^8 - 13x^7 + 15x^6 - 8x^5 + 12x^4 - 12x^3 - 10x^2 - 7x + 8$

1.6 encrypt(message, public_key, r = None)

This function performs encryption of a given message using a provided public key and returns $Z[x]$ encrypted message.

```
def encrypt(message, public_key, r = None):
    if r is None:
        r = generate_polynomial(d, d-1)
    return balancedmod(convolution(public_key,r) + message,q)
```

Example. Let's choose message $\mathbf{m} = -1 + x^3 - x^4 - x^8 + x^9 + x^{10}$ and random polynomial $\mathbf{r} = -1 + x^2 + x^3 + x^4 - x^5 - x^7$

$\mathbf{e} = \text{encrypt}(\text{message}, \text{public\_key}) = 19x^{10} + 6x^9 + 25x^8 + 7x^7 + 30x^6 + 16x^5 + 14x^4 + 24x^3 + 26x^2 + 11x + 14$

1.7 decrypt(encrypted_message, secret_key)

Let's decrypt the message. $\mathbf{a} = \mathbf{f} \cdot \mathbf{e} = p\mathbf{r} \cdot \mathbf{g} + \mathbf{f} \cdot \mathbf{m}$ (mod q). The next step will be to calculate $\mathbf{a}$ modulo p: $\mathbf{b} = \mathbf{a} = \mathbf{f} \cdot \mathbf{m}$ modulo p. $\mathbf{c} = \mathbf{f}_p \cdot \mathbf{b} = \mathbf{f}_p \cdot \mathbf{f} \cdot \mathbf{m} = \mathbf{m}$ (mod p). This function performs decryption of a given ciphertext using an own private key. Returns $Z[x]$ decrypted message

```
def decrypt(encrypted_message, secret_key):
    f,f_p = secret_key
    a = balancedmod(convolution(encrypted_message,f),q)
    return balancedmod(convolution(a,f_p),p)
```

Example.

$\mathbf{a} = \mathbf{f} \cdot \mathbf{e} = 25x^{10} + 29x^9 - 27x^8 + 7x^7 + 6x^6 + 7x^5 - 22x^4 - 11x^3 + 22x^2 - 7x + 3$

after central lift

$$\mathbf{b} = \mathbf{a} = x^{10} - x^9 + x^7 + x^5 - x^4 + x^3 + x^2 - x$$

$$\mathbf{c} = x^{10} + x^9 - x^8 - x^4 + x^3 - 1$$

We got the original message!

References

[1] Implementation by Elena Mashkina https://github.com/elena-mashkina/ntru/blob/master/NTRU.sage

[2] Explanation https://cr.yp.to/talks/2018.11.16/slides-djb-20181116-lattice-a4.pdf