NTRU Post-Quantum Encryption demo

Jakubovski Jevgeniy

January 30, 2022

To produce *.pdf from this LaTeX file (with SageTeX inside) from the command line on PC with Sage installed do the following:

```
latex example.tex
sage example.sagetex.sage
pdflatex example.tex
```

1 Example from Wikipedia

Let us try to re-create `https://en.wikipedia.org/wiki/NTRUEncrypt` We use the following common parameters:

```
N = 11; p = 3; q = 2^5
```

Select random polynomial:

$$f = -x^{10} + x^9 + x^6 - x^4 + x^2 + x - 1$$

Check that polynomials f_p and f_q with the property $f \cdot f_p = 1(\text{mod} p)$ and $f \cdot f_q = 1(\text{mod} q)$ exist.

1.1 balancedmod(f(x),q,N)

This is auxiliary helper function. It reduces every coefficient of a polynomial $f \in \mathbb{Z}[x]$ modulo q with additional balancing, so the result coefficients are integers in interval $[-q/2, +q/2]$. More specifically:

- for an odd q coefficients belong to $[-(q-1)/2, +(q-1)/2]$
- for an even q coefficients belong to $[-q/2, +q/2-1]$

Finally the resulting polynomial is fit into $\mathbb{Z}[x]$ and returned.

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

Example:

$$\text{balancedmod}(1 + 31x + 32x^2 + 33x^3 - x^4, 32) = -x^4 + x^3 - x + 1$$

1.2 multiply(f(x), g(x), N)

The following function performs multiplication operation specific for NTRU, which works like a traditional polynomial multiplication with additional reduction of the result by $x^N - 1$

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

1.3 invertmodprime(f(x),p,N)

This routine calculates an inversion of a polynomial modulo $x^N - 1$ and then modulo p with assumption that p is prime number. Returns a polynomial $f_p \in \mathbb{Z}[x]$ such as $f \cdot f_p = 1(\text{mod} p$. An exception is thrown if such polynomial $f_p \in \mathbb{Z}[x]$ does not exist. To find it we need functions find degree and compare degrees:

```
def findDegree(coefs_list):
    len_ = len(coefs_list)
    for i in range(len_ - 1, -1, -1):
        if coefs_list[i] != 0:
            return i

def compareDegrees(coefs_list1, coefs_list2):
    c1 = 0
    c2 = 0
    len1 = len(coefs_list1)
    len2 = len(coefs_list2)
    for i in range(len1 - 1, -1, -1):
        if coefs_list1[i] != 0:
            c1 = i
            break
    for i in range(len2 - 1, -1, -1):
        if coefs_list2[i] != 0:
            c2 = i
            break
    return c1 < c2
```

And now we can find inverse polynom

```
def invertmodprime(f,p):
    Zx.<x> = ZZ[]
    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

    if a.gcd(g) != 1:
        raise Exception("inversion dosen't exist!")
    while True:
        while list(f)[0] == 0:
            f /= Z
            c *= Z
            k += 1
        if findDegree(list(f)) == 0:
            b = 1/list(f)[0] * b
            res = Z^(N-k) * b
            return Zx(res.lift())
        if compareDegrees(list(f), list(g)):
            f, g = g, f
            b, c = c, b
        u = list(f)[0] * (1/list(g)[0])
        f -= u*g
        b -= u*c
```

Example:inverse for $-x^{10} + x^9 + x^6 - x^4 + x^2 + x - 1$ is

$$2x^9 + x^8 + 2x^7 + x^5 + 2x^4 + 2x^3 + 2x + 1$$

1.4 def invertmodpowerof2(f,q)

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 Zx polynomial h such as convolution of h f = 1 (mod q) raises an exception if such Zx polynomial h doesn't exist

```
def invertmodpowerof2(f, p):
    deg = int(math.log(p, 2))
    p = 2
    q = p
    b = invertmodprime(f, p)
    while q < p^deg:
        q = q^2
        b = b * (2 - f*b) % q % (x^N-1)
    b = b % p^deg % (x^N - 1)
    return b
```

Example:

$$f = -x^{10} + x^9 + x^6 - x^4 + x^2 + x - 1$$

inverse is

$$30x^{10} + 18x^9 + 20x^8 + 22x^7 + 16x^6 + 15x^5 + 4x^4 + 16x^3 + 6x^2 + 9x + 5$$

1.5 def validateparams()

checks params meet certain conditions: if q is considerably larger than p and if greatest common divider of p and q is 1 returns N, p, q

```
def validate_params():


    if q > p and gcd(p,q) == 1:
        return True
    return False
```

1.6 def generatepolynomial(d)

generates a random polynomial with d nonzero coefficients returns Zx polynomial

```
def generate_polynomial(d1, d2, N):

    result = [1]*d1 + [-1]*d2 + [0]*(N-d1-d2)
    shuffle(result)

    return Zx(result)
```

1.7 def generatekeys

generates a public and private key pair, based on provided parameters returns Zx public key and a secret key as a tuple of Zx f (private key) and ZxF_p

```
def generate_keys(polynomial_1 = None, polynomial_2= None):
    if validate_params():
        while True:
            try:
                if polynomial_1 is None or polynomial_2 is None:

                    f = generate_polynomial(d+1, d)
                    g = generate_polynomial(d, d)
                else:
                    f = polynomial_1
                    g = polynomial_2

                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("")
```

1.8 Encrypting

Public key is $pf_q * g$ mod(q)

```
public_key =-16*x^10-13*x^9+12*x^8-13*x^7+15*x^6-8*x^5+12*x^4-12*x^3 - 10*x^2 - 7*x + 8
def encrypt(message, public_key, r):
    return balancedmod(convolution(public_key,r) + message,q)
```

Example. Let's choose message $m = -1 + x^3 - x^4 - x^8 + x^9 + x^10$ $r = -1 + x^2 + x^3 + x^4 - x^5 - x^7$

$$e = 19x^{10} + 6x^9 + 25x^8 + 7x^7 + 30x^6 + 16x^5 + 14x^4 + 24x^3 + 26x^2 + 11x + 14$$

1.9 def decrypt(encryptedmessage, secretkey)

performs decryption of a given ciphertext using an own private key returns Zx 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.

```
encrypted_message=encrypt(-1+x^3-x^4-x^8+x^9+x^10,public_key,-1+x^2+x^3+x^4-x^5-x^7)
a = convolution(encrypted_message,f)
```

$$a = 25x^{10} + 29x^9 - 27x^8 + 7x^7 + 6x^6 + 7x^5 - 22x^4 - 11x^3 + 54x^2 - 7x - 61$$

The next step next step is to find b which is $x^{10} - x^9 + x^7 + x^5 - x^4 + x^3 - x - 1$

And the last step And we have the same message $x^{10} + x^9 - x^8 - x^4 + x^3 - 1$

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