NTRU Post-Quantum Encryption demo

Aleksey Mokhonko

January 28, 2022

1 Example from Wikipediaaa

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 $f \cdot f_p = 1(\mathrm{mod} p)$ and $f \cdot f_q = 1(\mathrm{mod} q)$ 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,N):
    g = list(((f[i] + q//2) % q) - q//2 for i in range(N))
    Zx.<x> = ZZ[]
    return Zx(g)
```

Example:

$$\mathrm{balancedmod}(1 + 31x + 32x^2 + 33x^3 - x^4, 32, 11) = -x^4 + x^3 - x + 1$$

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 multiply(f,g,N):
    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 $f \cdot f_p = 1(\mathrm{mod} p$. An exception is thrown if such polynomial $f_p \in \mathbb{Z}[x]$ does not exist.

```
def invertmodprime(f,p,N):
    Zx.<x> = ZZ[]
    # T is a quotient ring constructed from Z[x]
    # after its base being changed to Zp using ideal x^N-1.
    T = Zx.change_ring(Integers(p)).quotient(x^N-1)
    # Lift function converts Zp[x]/x^N-1 back into Zp[x].
    return Zx(lift(1 / T(f)))
```

Example:

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

$$f_p = \mathrm{invertmodprime(f, p, N)} = 2x^9 + x^8 + 2x^7 + x^5 + 2x^4 + 2x^3 + 2x + 1$$

Note that this is exactly the inverse mentioned in Wikipedia - NTRU. $-x^{10} + x^9 + x^6 - x^4 + x^2 + x - 1$

1.4 invertmodpowerof2(f(x), q, N)

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. It returns a polynomial $f_q(x) \in \mathbb{Z}[x]$ such as $f \cdot f_q = 1(\mathrm{mod} q)$. An exception is thrown if such polynomial $f_p \in \mathbb{Z}[x]$ does not exist.

```
def invertmodpowerof2(f,q,N):
    a=2
    b=invertmodprime(f, 2, N)
    while a<q:
        a=a^2
        b=multiply(b, (2-multiply(f,b,N)),N)
        Za.<z>=PolynomialRing(Integers(a))
        f1=Zx(Za(b))
    Zq.<x>=PolynomialRing(Integers(q))
    return Zx(Zq(b))
```

Example:

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

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

1.5 validateparams()

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

1.6 generatepolynomial(d,N)

```
def generatepolynomial(d,N):
    assert d <= N
    result = N*[0]
    for j in range(d):
        while True:
            r = randrange(N)
            if not result[r]: break
        result[r] = 1-2*randrange(2)
    return Zx(result)
```

1.7 generatekeys()

```
def generatekeys(N):
    if validateparams():
        while True:
            try:
                f = generatepolynomial(d,N)
                g = generatepolynomial(d,N)
                f_q = invertmodpowerof2(f,q)
                f_p = invertmodprime(f,p,N)
                break
            except:
                pass
        public_key = balancedmod(p * multiply(f_q,g,N),q)
        secret_key = f,f_p
        return public_key,secret_key
    else:
        print("Provided params are not correct. q and p should be co-prime, q should be
```

1.8 generatemessage()

```
def generatemessage():
    result = list(randrange(3) - 1 for j in range(N))
    return Zx(result)
```

1.9 encrypt(message, publickey,r,N)

```
def encrypt(message, publickey,r,N):
      Zq.<x>=PolynomialRing(Integers(q))
      return Zx(Zq(balancedmod(multiply(publickey,r,N) + message,q,N)))
      #return balancedmod(multiply(publickey,r,N) + message,q,N)
```

Example:

$$m = -x^{10} + x^9 - x^8 - x^4 + x^3 - 1$$

$$r = -x^7 - x^5 + x^4 + x^3 + x^2 - 1$$

$h = \mathrm{pf_q} * \mathrm{g} = -16x^{10} - 13x^9 + 12x^8 - 13x^7 + 15x^6 - 8x^5 + 12x^4 - 12x^3 - 10x^2 - 7x + 8$

$e = \mathrm{r} * \mathrm{h} + \mathrm{m} = 17x^{10} + 6x^9 + 25x^8 + 7x^7 + 30x^6 + 16x^5 + 14x^4 + 24x^3 + 26x^2 + 11x + 14$

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