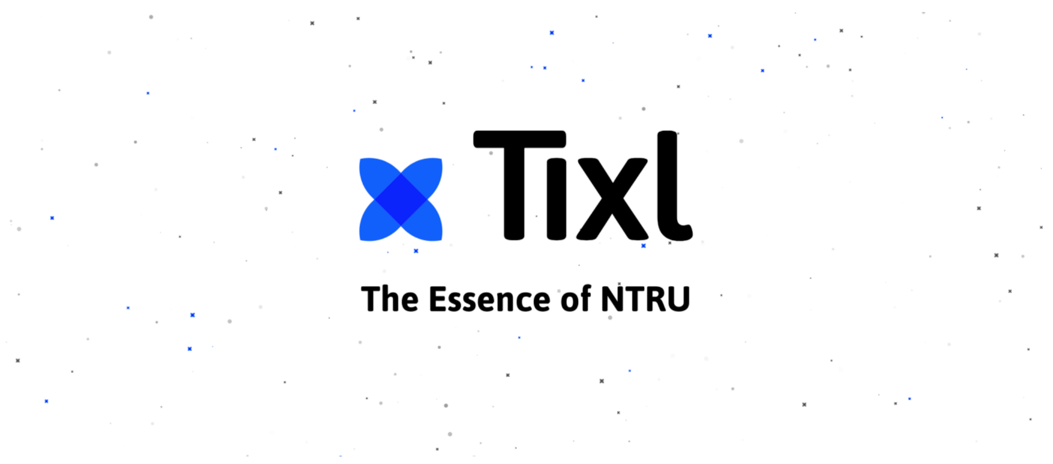
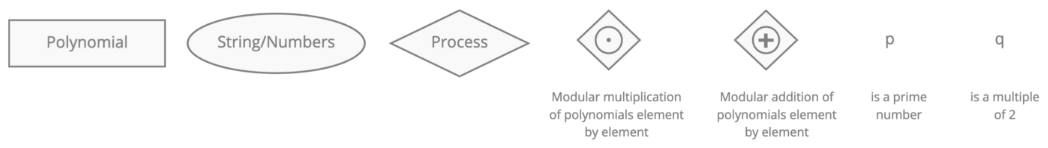
The Essence of NTRU: Key Generation, Encryption, Decryption

Vihren Stoev



As we have already established, the cryptosystem consists of three parts, **Key Generation**, **Encryption** and **Decryption**.


Polynomial
String/Numbers
Process
Modular multiplication of polynomials element by element
Modular addition of polynomials element by element
p
is a prime number
q
is a multiple of 2

Legend for this article

Key Generation

First Bob creates the keys (public key for everyone and private key for himself) and releases the public key to the public; this process is called **Key Generation**. So how does Bob go about generating those keys? First of all, the general parameters of the system, *N*, *p* and *q* are decided by either Bob or by all the participants in the exchange. They are not secret in any way and anyone can know them. After deciding on a nice and secure combination of *N*, *p* and *q* (*N*=251 *q*=128 *p*=3 are considered standard security level, for example), Bob chooses two random polynomials *f* and *g*, with a degree at most *N*-1 and integer coefficients between [-1 ; 1] (which means the coefficients are the numbers -1, 0 or 1). An example of such a polynomial is $-x^3-x^2+x-1$. The polynomial *f* must satisfy the additional requirement, that it has inverses under modulus *p* and under modulus *q*. If *f* does not have those inverses, Bob must simply choose another polynomial *f*. Wait a minute, some of you might say — Modulus? Inverses? How do we calculate that? Patience, dear readers, all will be revealed and illustrated later.

Now Bob calculates the inverses *fp* of *f* (mod *p*) and *fq* of *f* mod *q*:

$$f*f_p = 1 \pmod{p} \text{ and } f*f_q=1 \pmod{q}$$

If any of those inverses does not exist, now Bob must find another *f* and repeat the process until it has inverses, that should be fairly easy, with rising *N* the chances of any random *f* not having an inverse become very tiny. Now Bob calculates the number:

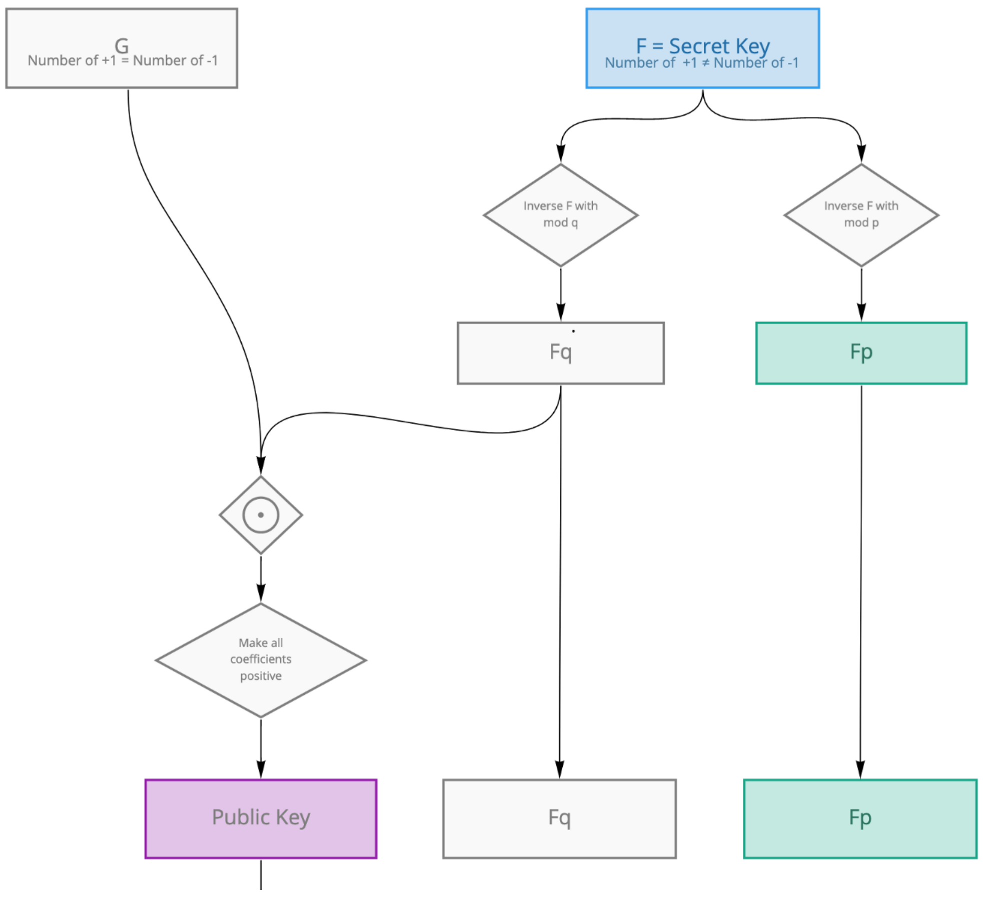
$$h=pf_q*g \pmod{q}$$

Here, *p* is, as we know — a number given from the beginning, *fq* is the inverse polynomial of *f* mod *q*, and *g* is another random polynomial without an inverse.

The number *h* is the public key which Bob releases to the world, including Alice. Sometimes, the numbers *N*, *p* and *q* are also considered a part of the public key, since they are also available to the public.

The polynomials *f*, *fp* and *g* are Bob's private key and he should keep them secure and private from anyone.

Now, with respect to encryption of the amount the polynomial *h* and the numbers, *p*, *q* and *N* are available to the general public (however they are not needed for the verification), while the polynomials *f*, *fp* and *g* are integrated into the account security and every account will use them to decrypt any incoming amounts.


G
Number of +1 = Number of -1
F = Secret Key
Number of +1 ≠ Number of -1
Inverse F with mod q
Inverse F with mod p
Fq
Fp
Make all coefficients positive
Public Key
Fq
Fp

NTRU Key Generation

Encryption

Now in order to encode a message, we have to turn it into a polynomial, so that it can be encrypted. In our case the encrypted message is the amount of currency (sent, received or left in the account), thus it is a number and can easily be turned into a polynomial; however for the cases of NTRU we need a polynomial with coefficients between -1 and 1. We can turn our number into such a polynomial by changing it into binary (or ternary system) so for example 13 can be represented as 1101 in binary and this becomes the polynomial $1*x^3+1*x^2+0*x+1 = x^3+x^2+1$.

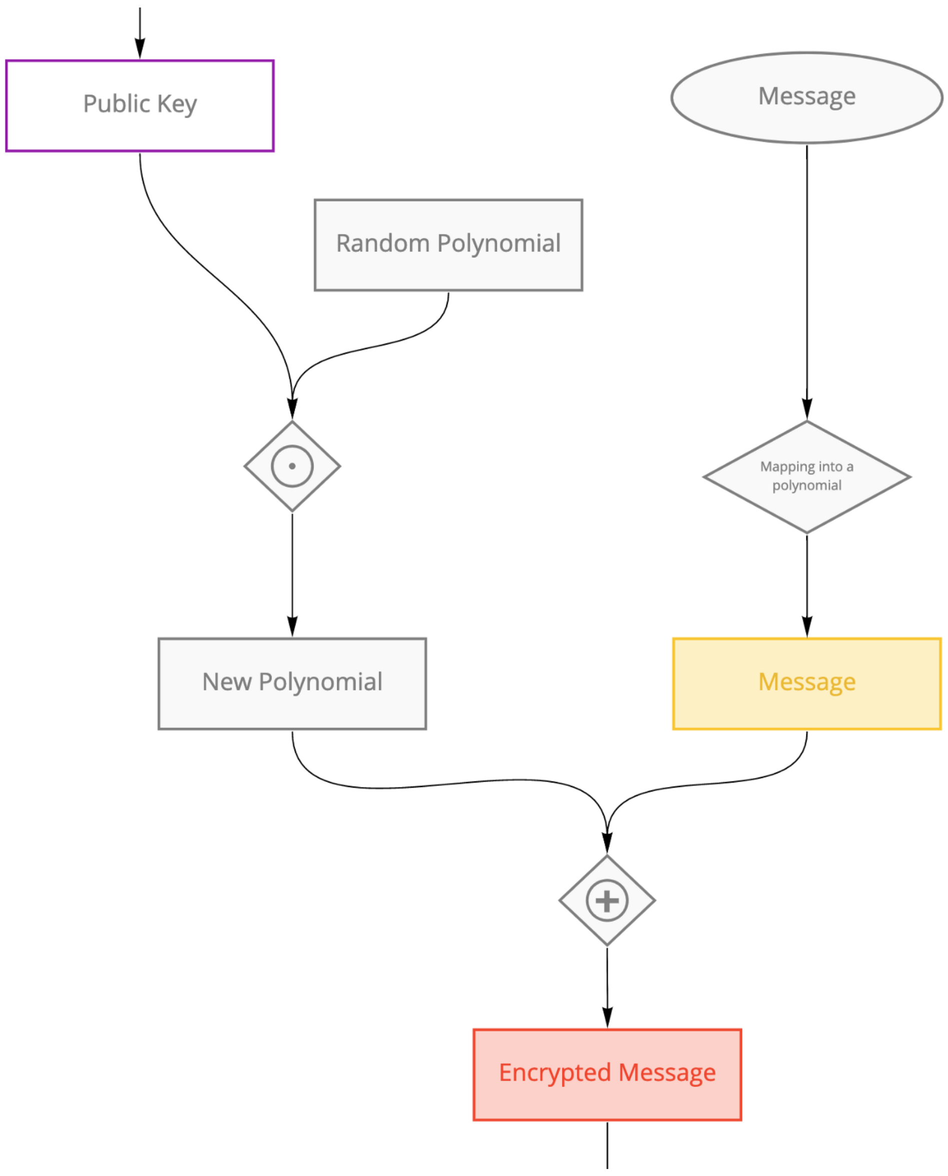
Now, after Alice has turned the message into a polynomial *m*, she is ready to use the public key *h*, that she acquired from Bob. Next, she chooses a random polynomial *r*, which is called the **Blinding Value**. This is the crucial moment of the encryption, since this unknown and arbitrary polynomial hides the polynomial *m* and makes it impossible to decrypt without knowledge of the private key. (Note that this blinding value is different in any encryption or, in our case — in any transaction).

Now Alice calculates the ciphertext *e (Encrypted Message)*

$$e = r^*h + m \ \ (\text{mod } q)$$

Here *e* is the ciphertext that Alice gets, *r* is her randomly chosen polynomial, *h* is the public key that she got from Bob, *m* is the message in polynomial form and *q* is the number that everyone knows as a parameter of the system.

Now Alice can send her ciphertext to Bob. In our specific case, for example, this ciphertext is used to encrypt transaction amounts.



NTRU Encryption

Decryption

Bob has received Alice's ciphertext *e*. Now Bob can use his private key to decrypt it. The first step of the decryption is calculating a polynomial *a*, such that

$$a = f * e \pmod{q}$$

Where *f* is part of Bob's private key, *e* is the ciphertext received from Alice, *q* is the well-known parameter of the cryptosystem

The coefficients of a are chosen to be between –*q*/2 and *q*/2 to avoid decryption problems, they should generally lie in an interval of length *q*.

Now, Bob proceeds to the next step, which is calculating a polynomial *b*, such that:

$$b = a \pmod{p}$$

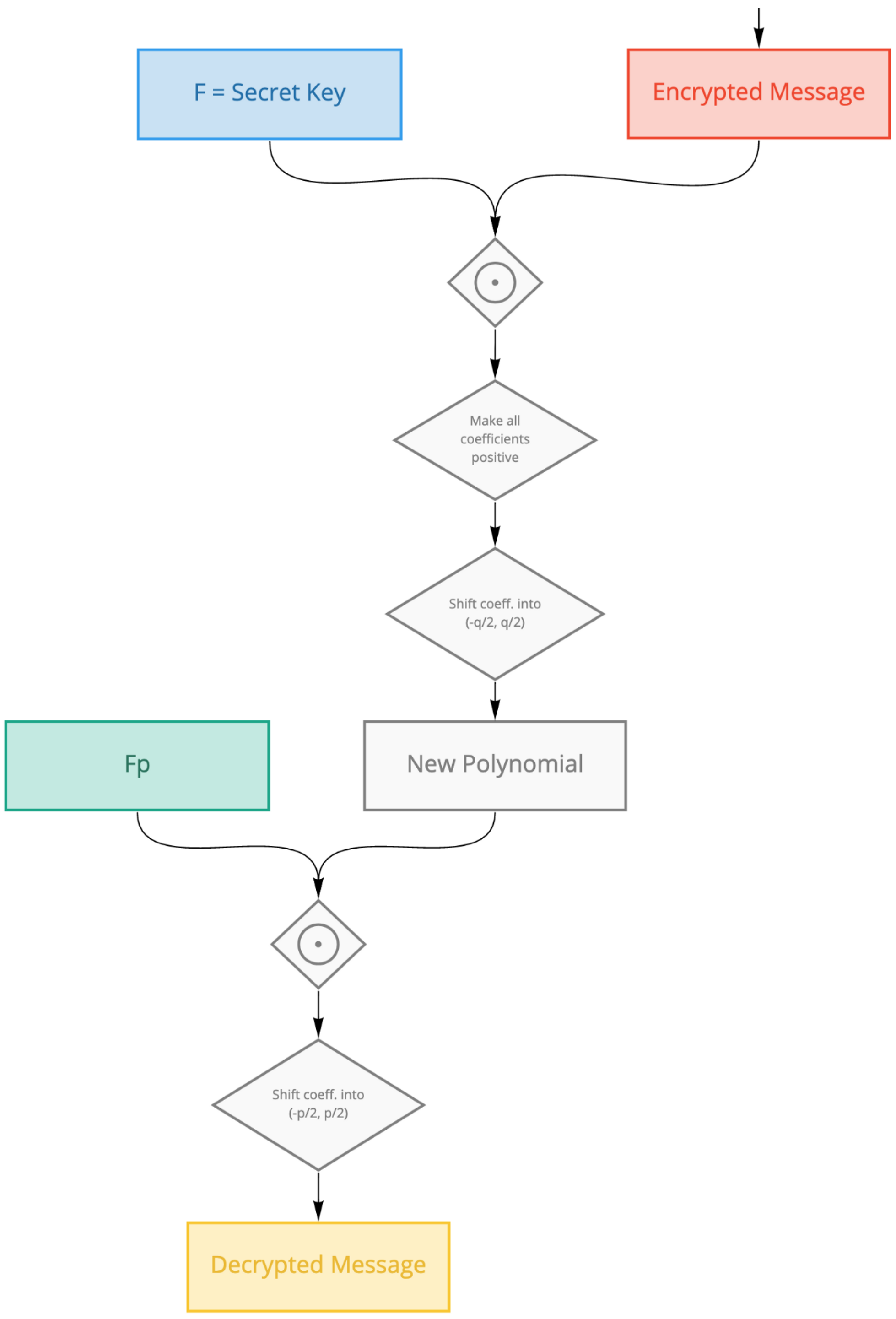
Here *a* is the polynomial Bob just calculated and *p* is one of the parameters of our NTRU

Then finally, Bob uses his other secret polynomial *fp* to obtain the message:

$$m = f_p{}^{*}\, b \;(\mathrm{mod}\; p)$$

Here *b* is the polynomial Bob just calculated, *fp* is part of the private key and the inverse of the polynomial *f* that Bob chose randomly and *p* is one of the parameters of NTRU

And *m* is the message he received from Alice.


F = Secret Key
Encrypted Message
Make all coefficients positive
Shift coeff. into (-q/2, q/2)
Fp
New Polynomial
Shift coeff. into (-p/2, p/2)
Decrypted Message

NTRU Decryption

But what about the related mathematics?

Now in order to understand how the encryption, decryption and key-generation processes work, we shall present some basic mathematical concepts that are used in the NTRU cryptosystem.

Modular arithmetic

Since everything in the NTRU cryptosystem happens under a modulus, it is very important to see how modular arithmetics work. In very broad terms modular arithmetic is simply division where you keep only the remainder and everything else just goes away. For example, the expression 123 (modulo 10) means to divide 123 by 10 and keep the remainder. Now 123 divided by 10 results in the number 12 and a remainder of 3 (since 123 = 10 *12 + 3), so 123 (modulo 10) is equal to 3. This is written as an equality (called a congruence) 123 = 3 (modulo 10), but the word “modulo” is usually shortened to just “mod” and the congruence becomes:

$$123 = 3 \pmod{10}$$

In general, the expression *a* (mod *m*) means to divide a by m and keep the remainder. Similarly, a congruence *a* = *b* (mod *m*) simply means that *a* and *b* leave the same remainder when they are divided by *m*. This is the same as saying that the difference *a* — *b* is a multiple of *m*. The integer *m* is called the modulus of the congruence.

Numbers and congruences with the same modulus may be added, subtracted, and multiplied just as is done with ordinary equations. For example:

$$(3 \bmod 25) + (9 \bmod 25) = 12 \bmod 25$$
$$\text{and}$$
$$(3 \bmod 25) * (9 \bmod 25) = 27 \bmod 25 = 2 \bmod 25$$

If *a* and *m* have no common factors, then it is also possible to find an inverse for *a* (mod *m*), that is, to find an integer *b* so that *a***b* = 1 (mod *m*).

For example, the inverse of 10 (mod 23) is 7, since 7*10=70=1 (mod 23).

The Ancient Greek mathematician Euclid is credited with the discovery of a quick algorithm, called the Euclidean algorithm, which can be used to check if a and m have common factors and also to compute the inverse of *a* (mod *m*) if they do not have common factors.

Arithmetics in the Truncated Polynomial Ring

As we have established, the principal objects of the NTRU encryption are the polynomials of the form $a_0 + a_1x + a_2x^2 + \ldots an\text{-}1x^{N-1}$ and the set of all those polynomials is denoted by R. Arithmetics in R follows some general rules, very similar to our general definition of arithmetics, however there is a very important addendum to the multiplication part which comes from the fact that the ring is of all polynomials of degree at most N-1 and not of all polynomials ever. So, first of all, let’s define addition and subtraction. They work exactly as one would expect:

If you have two polynomials *a* and *b*, such that:

$$a = a_0 + a_1x + a_2x^2 + \ldots a_{n-1}x^{N-1}$$
$$\text{and}$$
$$b = b_0 + b_1x + b_2x^2 + \ldots b_{n-1}x^{N-1}$$
$$\text{then}$$
$$a+b = a_0 + b_0 + (a_1 + b_1)x + (a_2 + b_2)x^2 + \ldots(a_{n-1} + b_{n-1})x^{N-1}$$

Now, the tricky bit comes in the multiplication. If we do the multiplication the normal way, we will probably get some values for x^N, x^{N+1} etc. After doing the multiplication, the power x^N is to be replaced by 1, the power x^{N+1} is to be replaced by x, the power x^{N+2} is to be replaced by x^2 and so on until we have finished with all values of x with a power higher than N. This property is what turns the set of polynomials R into the mathematical term Ring. Mathematically, this is expressed by:

$$a*b = c_0 + c_1x + c_2x^2 + \ldots c_{n-1}x^{N-1}$$

where the *k*-th coefficient *ck* is given by

$$c_k = a_0b_k + a_1b_{k-1} + \ldots + a_kb_0 + a_{k+1}b_{N-1} + a_{k+1}b_{N-2} + \ldots a_{N-1}b_{k+1}$$

This is better illustrated by an example. Suppose N=3 and our polynomials are:

$$a = 5+3x+2x^2$$
and
$$b = 1-2x+x^2$$
then
$$a + b = 5 + 3x + 2x^2 + 1 - 2x + x^2 = 5 + 1 + (3-2)x + (2+1)x^2 =$$
$$= 6 + x + 3x^2$$

$$\text{and } a * b = (5+3x+2x^2) * (1-2x+x^2) =$$
$$= 5 - 10x + 5x^2 + 3x - 6x^2 + 3x^3 + 2x^2 - 4x^3 + 2x^4 =$$
$$= 5 - 7x + x^2 - x^3 + 2x^4$$

and here we replace x^N with 1 and x^{N+1} with x, thus $x^3 = 1$ and $x^4 = x$ so we have

$$a * b = 5 - 7x + x^2 - x^3 + 2x^4 =$$
$$5 - 7x + x^2 - 1 + 2x =$$
$$= 4 - 5x + x^2$$

With addition and multiplication defined in this way, all their normal properties are true, for example the commutative, associative and distributive law:

$$a + b = b + a$$
$$a * b = b * a$$
$$(a + b) + c = a + (b + c)$$
$$(a * b) * c = a * (b * c)$$
$$a * (b + c) = a * b + a * c$$

And as things go with modular arithmetics, when we have calculations with polynomials under a modulus *q*, we divide all coefficients by *q* and take only the remainder. For example:

$$13 + 6x + 9x^2 + 12x^3 \pmod 4) = 1 + 2x + x^2$$

Inverses under the modulus

In NTRU we would often need to find the inverse of a polynomial under a certain modulus. The inverse of a polynomial *a* mod *q* is a polynomial *aq* with the property that *a***aq* =1 (mod *q*). Not every polynomial has an inverse modulo *q*, but most of them have and there are several algorithms designed to find whether the polynomial has an inverse into which we would not go right now. For example, if we take:

$$N=7,\ q=11, \text{ and } a=3+2x^2-3x^4+x^6$$

$$\text{Then, the inverse } a_q = -2+4x+2x^2+4x^3-4x^4+2x^5-2x^6,$$

$$\text{since}$$

$$a * a_q = (3+2x^2-3x^4+x^6) * (-2+4x+2x^2+4x^3-4x^4+2x^5-2x^6) =$$

$$= 12 + 22^x - 22x^3 + 22x^6 =$$

$$= 1 \pmod{11}$$

Those inverses are found more easily with a version of the Euclidean algorithm, mentioned above.

But Why does it Work?

By now probably many of you are tired and confused by all this math. Don’t worry, the hard part is done. In the next Cryptography blogpost, we shall explain why it works, have a nice example to show how everything we saw today works and then see where can we go from here.