NTRU Algorithm Specification

Done By:
Sundar R

Prerequisite

We will require some basics of Abstract Algebra

Example: Ring, space

What is a Ring??

A ring is an algebraic structure consisting of a set equipped with two binary operations: addition and multiplication.

In a ring, addition and multiplication are associative, addition is commutative, and multiplication distributes over addition. Rings also have an additive identity (0) and an additive inverse (-a) for every element a.

Example:

Integers are good examples of a RING

What is a Space??

a **space** refers to a set of elements that share common properties and are subject to certain operations.

Ex: {2,4,6}

Secret Key vs Public Key vs Private Key

Secret Key: (Symmetric key)

Same key is used in both encryption and decryption, and generally the algorithms involving secret keys are much faster

Public Key: (Asymmetric key)

A key that is used only in encryption and is known to the outside world.

Private Key: (Asymmetric key)

A key that is used only in decryption and is known to only the receiver for whom message is meant for.

NOTE: Both Public and Private key work in pair and algorithm which involves them is often slower and complex.

Rapid Recap of Last Presentation

P and Q are just two coprime numbers. Specifically, P should be prime and Q is much larger than P. Q should be a power of 2.

If you want an algorithm I can give it in an exact form pls find it below:

f.fp=1(mod p)

f.fq=1(mod q)

Public key:h = p.fq.g

Private key:f,fp

Message:m

Random small polynomial:r

Cipher text e=r.h+m

a= f.e(mod q) = f(r.h+m) modq = f(r.p.fq.g+m) modq = r.p.g +f.m mod q

b= a(mod p) = (r.p.g + f.m)mod p = f.m mod p

c= b.fp(mod p) = f.m.fp(mod p) = m (mod p)

Introduction

We will be focusing about the key encapsulation mechanism(KEM) of NTRUHRSS

This is the latest NTRU submission in NIST

What is NTRUHRSS?

NTRU Hybrid **R**ing **S**ampling **S**cheme

It involves a fixed relationship between n,p and q we will see the values soon

After performing lots of experiments the author claims that NTRUHRSS701 version is the best to work with as it has shown good extent of protection against cyber threats

It is also named after the parameter chosen, So let see them.

Definitions:

- A ternary polynomial is one where all coefficients are either -1, 0, or 1.
- T is the set of all non-zero ternary polynomials with degree less than n-1.
- T+ is a subset of T, containing only polynomials with the non-negative correlation property.

Parameters:

- n is prime preferred n = 701
- p = 3
- q = 8192
- Lf = T+
- Lg = $\{X \cdot v : v \in T+\}$
- Lr = T ,
- Lm = T , and
- Lift (m) = p · S3(m).

Here S3 means its restricts the message to have coefficients in the set {-1,0,1} only

For ex: If $m = -2x^2+2x+2$
then $S3(m) = -x^2 +x +1$

Here Lf,Lg,Lr,Lm all are spaces from which the polynomials are chosen to work on

For example:

Assuming $N=3$

A polynomial f might be $1+x+x^2$

A polynomial g might be $(x+1)(x+1) = x^2+2x+1$

r and m might be $-x^2+1+x+x^2$

What is the motive of the paper??

The paper implements a cryptographic algorithm called NTRUEncrypt, which is used for key encapsulation. Key encapsulation is a method of securely transmitting Secret key over untrusted channels.

There are basically 3 main steps

1. Key Generation
2. Key Encryption
3. Key Decryption

I will explain all of these in the coming up slides

Key Pair Generation

What does a seed mean??

In the context of cryptographic operations and random number generation, the term "seed" refers to an initial value used to start the process of generating random numbers or cryptographic keys. It's like a starting point or an initial input that, when combined with an algorithm, produces a series of pseudo-random numbers or other cryptographic values.

So steps to followed are:

1.Declare a seed array[just a space to store values]

2.Generate a random seed using randombytes function

3.Then from this seed generate random key pair (public key and secret key)

4.Randomize it little more by appending the secret key with additional random bytes.

What is it about randombytes function??

The `randombytes` function it is present in <cstdlib.h>. It can be thought of like a machine that shakes numbers around to make them unpredictable. It uses special techniques (like AES256_ECB) to create random numbers that are essential for keeping secrets safe in computer programs, like when you're sending secure messages or protecting passwords

Now what is AES256_ECB??

The `AES256_ECB(DRBG_ctx.Key, DRBG_ctx.V, block)` takes a secret key and a value, encrypts the value using the key, and stores the result in a block. This encrypted block is used to generate secure random numbers in the program.

AES stands for Advanced Encryption Standard, which is a method for encrypting (or scrambling) data to keep it secure.

256 means it uses a 256-bit key for encryption, making it very secure.

ECB stands for Electronic Codebook mode, which is one way to apply the AES encryption to data.

<u>How are pk and sk populated??</u>

f and g are generated through by sample_iid function

- It generates a function whose coefficients is {0,1,2} and checks <x·f,f> > 0

<x·f,f> > 0 what does this mean??

x.f is cyclic shift

For ex: if N=3

Then f = {1,0,2} then x.f = {2,1,0} now <x·f,f> this means just dot product of the both the values here 1*2 + 0*1 + 2*0 = 2>0 so this polynomial meets the requirement.If it doesn’t hold true we have to invert the sign of even indexes

Now inverse of f(with respect to p) is computed as one private key and f itself is a private key

Similarly g*f is calculated and stored as public key.

Now we have our public key and private key stored in the character array dedicated of them

Key Encryption

Algorithm involved is:

1. Generate a Random Seed for Message Encoding
2. Using this seed create two random polynomial r and m
3. Convert this polynomial into byte array say rm
4. Hash the byte array to derive the shared secret k
5. Convert Polynomial `r` and Encrypt Using Public Key `pk`

What is a byte array??

A byte array is a data structure that stores a sequence of bytes, which are 8-bit units of data.
This conversion is necessary to prepare the data for hashing and encryption operations

So how does rm look like??

rm: [r_byte_1, r_byte_2, ..., r_byte_N, m_byte_1, m_byte_2, ..., m_byte_M]
Since r and m are generated in such a way that its coefficients are in the set {-1,0,1} so rm polynomial also has coefficients in the set {-1,0,1}

What is Hashing??

Hashing is a process used to transform a given input (or "message") into a fixed-size string of bytes.

- Here we use crypto_hash_sha3256 function to hash the rm byte array

What is special about crypto_hash_sha3256 function??

It implements the SHA-3 hash function and outputs a 256-bit (32-byte) shared secret k

What is this SHA-3 function all about??

- SHA-3 is a family of cryotograhic hash function and keecak algorithm is the core idea behind it
- It includes functions to absorb input data into the state, permute the state, and squeeze out the final hash output.

What is a state??

- **state** refers to a specific representation of data that is used and modified during the computation of the algorithm.
- For example, in the Keccak (SHA-3) algorithm, the state is a 1600-bit (200-byte) array that is repeatedly transformed through a series of permutations and bitwise operations to absorb the input data and produce the final hash output.

Hashing involves four stages:

- State Setup: the state is represented as uint64_t s[25];
- Absorbing input:During this phase, input data is XORed into the state. `keccak_absorb` function updates and modifies it content
- Permutation: `KeccakF1600_StatePermute` is called,it mixes the state bits in a complex way using bitwise operations.
- Squeezing output:`keccak_squeezeblocks` is called, it further permutes the state and extracts the output. The output that we get becomes the shared secret.

Author claims that these are just helpers functions that can be called when ever you want as they are available in open source.

So wants us to focus on core logic than these side functions.

Lets walk through an example

$h(x) = 2+3x+4x^2$

Compute ct = r.h mod(q,x^n-1)

In this case assume $r(x)=1+x-x^2$ and n=5,q=7,p=3(lifting)

$m(x)=1+x$ and $h(x) = 2+3x+x^2+4x^3+5x^4$

$ct(x) = (1 + X - X^2) * (2 + 3X + X^2 + 4X^3 + 5X^4) = 2 + 3X + X^2 + 4X^3 + 5X^4 + 2X + 3X^2 + X^3 + 4X^4 + 5X^5 - 2X^2 - 3X^3 - X^4 - 4X^5 - 5X^6$ Reducing modulo $(X^5 - 1)$ and then modulo 7: $\equiv 2 + 5X + 2X^2 + 2X^3 + 5X^4 \pmod{7, X^5 - 1}$

Lift the message m: liftm = p * m = 3 * (1 + X) = 3 + 3X

Add the lifted message to ct(x): $ct + liftm = (2 + 5X + 2X^2 + 2X^3 + 5X^4) + (3 + 3X) \equiv 5 + X + 2X^2 + 2X^3 + 5X^4 \pmod 7$

So finally $ct(x) = 5 + X + 2X^2 + 2X^3 + 5X^4$

Key Decryption

These are the steps involved in decryption

1. Decrypt the ciphertext `c` using the secret key `sk`
2. Hash `rm` to derive the shared secret `k`
3. `Concatenate secret PRF key and ciphertext for further hashing`
4. Hash the concatenated buffer to derive `k`
5. Conditional move to set `k` to 0 if decryption failed

Lets continue with our example

We know that

Encryption result (ciphertext): c(X) = 5 + X + 2X^2 + 2X^3 + 5X^4

Secret key components:

f(X) = 1 - X + X^2 (private key polynomial)

f_p(X) = 1 + X - X^2 (inverse of f modulo p and X^N - 1)

h_q(X) = 2 + 4X + 6X^2 + X^3 + 5X^4 (inverse of f modulo q and X^N - 1)

Decryption process:

1. Compute c * f (mod q, X^N - 1): (5 + X + 2X^2 + 2X^3 + 5X^4) * (1 - X + X^2) ≡ 5 + 4X + 0X^2 + 0X^3 + 5X^4 (mod 7, X^5 - 1)
2. Reduce the result modulo p (= 3): 5 + 4X + 0X^2 + 0X^3 + 5X^4 (mod 3) ≡ 2 + X + 0X^2 + 0X^3 + 2X^4 (mod 3)
3. Multiply by f_p (mod p, X^N - 1): (2 + X + 0X^2 + 0X^3 + 2X^4) * (1 + X - X^2) ≡ 1 + X + 0X^2 + 0X^3 + 0X^4 (mod 3, X^5 - 1)

The result 1 + X is our original message m(X).

4. To recover r(X), compute: b(X) = c(X) - Lift(m(X)) (mod q, X^N - 1) = (5 + X + 2X^2 + 2X^3 + 5X^4) - (3 + 3X) ≡ 2 + 5X + 2X^2 + 2X^3 + 5X^4 (mod 7, X^5 - 1)
5. Then compute r(X) = b(X) * h_q(X) (mod q, X^N - 1): (2 + 5X + 2X^2 + 2X^3 + 5X^4) * (2 + 4X + 6X^2 + X^3 + 5X^4) ≡ 1 + X + 6X^2 + 0X^3 + 0X^4 (mod 7, X^5 - 1)
6. Reduce r(X) modulo p: 1 + X + 6X^2 + 0X^3 + 0X^4 (mod 3) ≡ 1 + X - X^2 + 0X^3 + 0X^4 (mod 3)

We recover r(X) = 1 + X - X^2, which is our original r polynomial.

Now since r and m is recovered we can make rm out of it.

Now we have rm with us. What next??

- The decrypted message `rm` is hashed using a hash function (SHA-3-256)
- If decryption was sucessful then we maintain a variable `fail` which is set to zero and if there was any issues in decryption process we set `fail` as some non zero value.
- The result of this hashing is stored in k which becomes the shared secret
- ***We are done with this but author prescribes following steps for more security***
- Now we prepare a buffer space will hold a concatenation of:
    A. Part of the secret key (Depends on implementation but here its (N/5 + 1));
    B. The ciphertext `c`
- Now again we use the same hash function on this buffer and update the new value to rm
- Now this step has to be done to secure the shared secret(k)
  If `fail` != 0 we immediately replace the k with rm so that shared secret is not visible to the outside world.

Some Major questions to be addressed

Why double hashing??

- After doing continuous experiments, author proclaims that single layer of hashing exposed a good chance for message and secret key to be cracked open using different techniques like deliberate tampering of the Message Authentication Codes etc.

- So 2nd layer of hashing was required to make it foolproof

Why after message is decrypted the process doesn't stop??

- Main motive of paper wasn’t to decrypt the encrypted message,of course it does happen eventually but the main aim was to have a Separate Secure Communication channel

- This Separate channel makes the recipients use even symmetric key and thus make the communication much faster and smoother.

Interested in seeing output??


```
seed = 061550234D158C5EC95595FE04EF7A25767F2E24CC2BC479D09D86DC9ABCFDE7056A8C266F9EF97ED08541DBD2E1FFA1
pk = 54050CF5C4E5AAB6CD62C2EBD092AEF03A2FE5521BDF836E5197F23F22F1925BC3BD6C35413983E77DB48A80CB52AC9403F72AE8B10B66CBEEF480B04409B5B6C67D8EAC5DB57F5FAE2F20914B7CB4FB6BD471B20D781863F39A71B115
5E6332D3B3415A7869DAD31CEAD13036FC03D84007B39F52507DD659D4F2A9B45C55A4B27591CACFBE7C0E52943AE7864C10F76681C6D31555E6F95980EF5B3E408E022ED406C4860E4B6710C286C5867776F5EFABB4E4AF3041E841C2A3261
1E219868C7715D08D91AE6E30D1A0CBA8F74712FE05FFEA26DE995DF9C03061B4592D2EFFF3846FA64B35CADA90667C879E7BD961AEAB1103D2AE3E9D4310353435C95EB943F2A42D4A40D08C6A6DB8E5714196247CF48C2619E9791E7BD0C3
3900DB4BEE1A7E3614DA0A510933BB8C5FBCC8C395F505160B67B4B09C2D36F4430297E26B50B2E4005FC2A43B4A3924539067931B16DEBE7251D0EC27B8825F5C19E296C53AFB4F276DB6B451A0E67A608DCBB0661B818B7DCE6EE24E7EE64
B29CE81D5F5D8A90EBA0AC2A2A14F05A79D249D5F833FC50DB577244E7F9DB1C38515A4FA8A6176666AFFA5EED3301252CF2BB4486AABE016EDC863120622960E57276E221FAB7A5247E91276C1063B82FC07E4DA91E52551438D1CD1FC2BA9
BB7D8E73FE1FD592D40334FA1C3F6E06DBDD56848C50B5BD7A8185FBBFE9814CE06A59EEB379657597C701D0BF5F0EC52928B489BB3D57204EF4D4F4769F5ED39F5B9397F5A4A179D1AF0B11463B91CC177A6743C7027B1EF957C36E90D8A1D
FFBF55AA2C60A02EFE5B5669EEC7DC98345D8DA3851C77E8AD37BF8F54E260E89FADB4385317EBE42DE72D4FEBB64FE40CAF55E2B9B4E82930FA05044129E416AC0224ADD03B4408C50501AB7F7779A461ABF17239CE949445A3C754E7DB96E
2A1AE72F7D884F9B5131D55E1F4B1CEDC05A157795E9F69D7B86793C5B1E085DDD7DA065BF9BB22BDCAD305267AC28F510F3EBDDEA656994058D8E7E3902C467DFA112370C37CEE77E7A130EDDC454212B38DFB4A7017240F78B23C9A90A2C9
83661FB03AF46F6D6258841E5568B4F521ACE028019296A738A1FEF595D5B16728EC96571A14AF4FBE593B837C05E9BACA031EFD3353D0A62574F7CB7DF79418E866D72B3D3015C0173B7EAAC996F35320716897518C9F5AC7339534192009A
BF73CC11351216D7310E118CE4B8D2FF03B49157A52FED5572DA34D1E1B6C52D942C083710A46880D8D96AD1E1CEA6B26543C09E194BFD4D24D89F36B343BEE70ED8021C5D47A1F9A447062FD6966A461C3DC008CBC738015A90B0FFAA05D4A
D6A5E3EDDD0A2718606E11ED808EBE28C9FD1BAB46F3A7661CC508C5754EAEAB3941681615CDECE45FE9EB1A3E03B2C9490C187201F267B8BC629618CBF90E91A95FA15D94484356A58F8D596DAC853B1DDE7211F18C4986CA9DCF4CED62362
C22C792CAF9ECFA8E5B786DAA81E5E9E8DCD51F9D2ABA3EBF8E0DF1A78BFDFF7338538B39DF6FBABE0B6271E2351B92672A9476F09EDDC879261BC0F434DFA3BDE9046DF531FC3CBF15484FE3807E4B059EAF326578C3BBDE905
sk = D067D98F0055E2C3DEEF1076BBB755AFD065112C85C46E6C350655D9625073E94E4528C053E720206CDB780A61AD5120C498A4CD3E60C334D8702F0F79D81418AF959CAB7722EBDD30DF0A479E1403D337AD923E5DBCED4EE04839020F
7751C075A4892AABBB0E001ABA95515E1B65C63C503D97E51EAFB9AC930F5A1DBCB7E9388B5EF1E75E893E1B5E4995E90F71691C40B00352A96CB9BE2E1D94423F897DED5FD466E84CB72290014981DE962EBF41B42BE1E8B8B7BE2EDFDB49A
558A681458196D3B3950AA73F4DE4E8089BD4E161146FD11651C955DB4661DFE0C18185EE2D7A25E24DBACA2EAD84541DAAAC3795BB1FD2536CCEBC572EA471845B669C336F0FE7055DBF16197F5195369446789F0CAF8A0E388FE6846E5F35
B0E309F96A932F05502FDFFECB3071CF7FAD853A05B24A715CEFCAD72E4C56EA03ED498C90873D99C3A80DA48DA2AA7024F7E4AF7DC9A6546DCD596DBC8F5A65D564E08A119743768EF84A799DCEB49DF243212B823E5F0C8539CA9F6004F05
6123330450E25D0A1C3CD3C8A4BF331F4E5AFA770B2F795412B472D04FFAC26F1E826E72CE336A295857A7E65EBB1077C74F36102FCEEF34552954AA831A3CA6EA4EDEE2264DB71047FBCD42200D2E973E01A92CDD71DFCD290D4E55079A952
051D6E790AA0A85661D504113FD294C967B848D72F58DBBEE67C279347BF03DB5DEC7B0CD1D444A852EA831B1855E9458557E6250A9BD4CE5F297EA469732A0CDC01842A38A8F89CBB9BAD2319A6EFA95FE472A7D87FADBD9538837B6D00CF1
23AAB07585E2A0A8EEC91AC7AE087592AC737695A7CC66A1CA199622D2CCF3C46D3B53D1B4D840A5B2231B4E7680FAEF7DB8776E02E5B496C22091D03EC07A44D9E910F591CA498BFCF02F792D03EEB55643A917689101161940D6B59F9943B
720366E3B05549CE7D8972DF1A4F3C63251DB61084FE9365FFEDCEB7CEDE9E277EBAA43ABE567F92C97A53452562F05F32A7EE347EA1600E0AB97E686D0EBAF0C258EF8C7632D6CCAAA5323DAC2ED8FDD54578904931A614CD4E8D3386C93A4
160C57C2F72B42EB2D35AA287F1824DA821F90C2042255FFFD38DB39983557996F8677396137963FD0306B5A8702714477BB3501F7558E66DF10907CA9410C5370C2A99075A7CB066D1AF78EE91D91DDA377B36AE2DFA265AF620436FA11332
BBD61DAC0324FB90BA58D74CE6299E29C04EBF2FE441554537047E35B3CA436118EF28638A32A30EC1ACEB256FC1BB84CEE8127C82A725D1E803B61956E50C4D6AE0DC53FCB5CD0685D66E1507BFF52FFDC8EE6D01BD2492CB44200195B269
48608F1EEA7271CAF8970D497444EA546377270171A534AFEE05D01E05DA2DEE25049C14492B92E72B8F89EBB1E435630B7440226EB3E8566284065123DC025A53B8E9899E41B636554828FF8A79B41043314C5457981A8886CE2F5B55DE205
82F36CCFD5A85E439228D69B1A927516BB054F3795BA2530941E4973F2D09DD3C73274F3AD046A1DCE4AF78EA5AA18B768368AA097E7D6DA65475754F975038803F9832E5A9A0FC7A7F74FA30E410A89933BC124C95B31657371E3BD6E5378B
6E8866A47D3F141ED57EEF1FF5AAA64DDB7D99AE3663A5448E9A5E3F1EC06BCD3245892D6C56CC0E76B0E7B360306EE9C127AABD143ADAA3860C42EF1AF592C968642C39C1DB8D3D0A030E882EFC5C30374F5CA68F19B325F88201ED3338FD8
E0A0F61D4913C0B3CB685D75F96D55E24B9769B06EB05D17EFBB84E3F550A92FB398127214D167E76639844D02E6CF8384007CD1B21B540D0FC37FC80D8488B3C4F28F7F77498D9E60690B733EE47E0D5428AF4FCEFD4739333DA1E94E3B7B9
4F6F1E5EF8D90A7BE0F91C379E9DF1561A34684F0F1B2EA6B1598CEB1B60C249CC88C4A115AFC06B1A3A7779E2F6AB70930817C7AC336728423F13F919172A1D6F82D2B09A264610F3DE04F056602BC1C377B0EB14193A1C6E6FC6DA8E709C0
B3B456F996337536DF40EE5AC8571FE7C2BC7A19
ct = 4FB29CE5CE753444A0B941D877305F9338ED3F4F30F5E8A0BC1F41AD9817B1528C9E86CCFF4E6F6BD364C1EBA2AFF7D48335E71FAC9BFE5B85E9FF19B3E60EAA3517E88913F41CD010492B9C060D89BADFA5257B1EA77F1E32AE2C2FAE
AAB587AD305FBB99961CE88D39DBA48D8BC598EF3DC8BFECE97E67301BCE081EF8EE521EF23FE70A98834B63E021B4161D1A9CD003C300F2E9BEF0FDD841915319436313 7F28810462D512161C82FEF9D5C11E88250B00BF51202E02341078B
40F89BBE5EA8BF0132C6EFBD75A2C27286DF14634425520C7BF6603A6B805D1B37490E2F15164B4D276C1B4DE297F397B480275B8E39908A785E5B49B296473DA62270C2D0A396DB1898D190A043D042D4FD7FF4EDB90466861B2C590091459
D022F9826316BCE8ED5A57769DB3EFB51BB2FDDC3C7882D3FC3860C338E1664B7CCD93ACCA5EB9342DBDF10B1D9B53D39EF8917D9CDB0F222F6BB2BCB7ED50982846F39FA486239E738E48E59081BDC46485CF64B8D3A0CD1D430D266B1E89D
AC0C00B8DE19D8A8901BC6505FCD7A384A8C04464298D9E27420C6D973177B6C00B8F4E28DF7B6750AF0B7F6697270806DA8066479182DDBC866517AA494589A3857B63A4004672BE2D52105282C752E0D55F1C5A29A2345BE51B391A6DD96F
4BFB34EF0A6E43014C6F712E3B53F5F60A5025024810CBDE012C2FFB9BF4864BEC3A6B1E08D6DACFBDFE54C9D875CE88F3488F2CFBBAFA3833406215167A86DD4A6BC0EFC2E4CB6F286FF4670F23CA8D8AB78F55000F1438AA2CA11927B97FD
59D77C44BBDA6B7AED96FB8F66A257A3D101F4CE024D98E9DF157C8978624C8387658543D9D9645B57B923DE2702D4FB100740A82BB21BD8B79F14F7955684E49C4EC7E6B72A2F1736FAA1FEC3BB3B31DDAFED95B1E50A713EC0050D5A95B94
89AACF77847B87D1CBFFAEA90F1AF0EE5E2AACB5D80E9BF142CAB92722B0ED72BE31835C9FD0CF160C1400516BDC81AF7CC710146840FAE056A7898FFEFFD819B64C3E2E946D2AB4380631D9978EDB610BE0918EDB7C314A6F9714E394DFDD2
DE7C788B0E6B01A57FB46D0CF27FBF6FD459FBFC7FEF840D0B863ED6EF1E0A1A93ED0CD2432C55E8CDAADF6CB31B1F1872BDE39037F5DBC94CCDE7A43ABBF8902291FCAF063D6C4C17052F63A06AD88AEDA5BBECEFB914DA4CAB60F33E9A7AC
89EAD9A0DFB03AFEEA456AF7558649133E77862F2876A3E21FB57AB2293FCAFEF6BF71E55308FAED720216BB66D5A89172E38587B1B35D07F8DBAA5470F726C361C300C7B9672AE9B9F2DD8E7A171E3807DB745DDE434AC32060CD27F446F8A
ADBB2C234975D42308CE37C637227FCF7977B0C0142772AAFC047678508086B5FEA059506DD6601E152DACD2CF235B72FFF07B50874F25312551B28F6DB1E92501E1E0459A82AD54DF1CC9067FFA316D31CCE6747643EF514AB6C046189D81E
042E7E8FC9F2D190090B1F447C8F8672365A0F52838A178537B0918B2A28CE1F3A207115631F5A9C53CEEC453306DC03AB078FB672BF9A8765364E70904F1ED2E5432F5E9B83C5C7B5A7DF47D3B5E2B1C015A4E6722ECF9C8A06
ss = 10AF7BA1D625B16172C5B80E2EE53AE9B7F3EDBE2E226F113EDE5A0EA8D1A978
```

<u>Aren't we discussing about just polynomial operation with integers how is alphanumeric appearing in output??</u>

Yes,

We are dealing with just integers but the alphanumeric plain text appears because the random function generates every integers as hexadecimal number(base 16)

So we are able to see all numbers from 1-9 and A-F(all numbers from 1 to 15)

Here

A-10	D-13
B-11	E-14
C-12	F-15

Challenges I faced in this Internship

- My biggest challenge was to read research paper and understand the documentation
- The documentation by authors in some places like random seed generation or hash function calling wasn't explained at all.Couldn't get any resources also to work on them
- The README file literally told me that some functions like these exist call them properly with respect to their arguments
- Compatibility issue, the project was built with linux machine. Mine is Mac OS, so many libraries weren't of same names.Even path errors like frontslash and backslash difference in both OS.
- I couldn't get a user friendly algorithm to make polynomial f dynamic in naive NTRU. Of course I got them here but I don't want use very probabilistic algorithm for such naive implementation.I want a proper implementation

What did I learn and how did I overcome some of them??

- Iam actually very new to Bit Masking,I tried my level best to understand the hashing and random generation. Since documentation wasn't there it was very time consuming process to even get some resources for it.

- Here and there I did get some vague idea on what is happening from GPT,reditt, stack overflow and google but wasn't sufficient enough. Since I was in time crunch,I didn't care much about it because the documentation in itself focuses more on "owcpa.c" and "kem.c" than any other helper functions. So focused more on them.

- Understood "***patience is very much required for debugging***😅" as I wrote my own MAKE file from scratch for building the project on my system.Found out the way in which absolute and relative path is written for a file is different for different OS.

- Author committed some error in calling the functions so had to sit and debug by correcting the arguments.It was fun experience overall.

Future works

Performance optimization:

- Exploring more efficient algorithms for hashing and random seed geneartion as there is some grey area around it.
- Implementing and benchmarking different sampling methods for f and g polynomials.

Parameter selection:

- Conducting a comprehensive analysis to determine optimal parameter sets.
- Exploring the trade-offs between security, performance, and key/ciphertext sizes.

Want to clone my project on your system??

Project Github link is right over here 👇

https://github.com/sundar2k22/NTRU_NIST.git

References

1. https://ntru.org/index.shtml
2. https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf
3. https://en.wikipedia.org/wiki/NTRUEncrypt#Public_key_generation
4. https://en.wikipedia.org/wiki/Euclidean_algorithm
5. https://asecuritysite.com/lattice/ntru_key?N=11&p=7&q=97&f=-1%2C1%2C1%2C0%2C-1%2C0%2C1%2C0%2C0%2C1%2C-1&g=-1%2C0%2C1%2C1%2C0%2C1%2C0%2C0%2C-1%2C0%2C-1
6. https://nitaj.users.lmno.cnrs.fr/ntru3final.pdf
7. https://www.ijcaonline.org/research/volume124/number7/zalekian-2015-ijca-905527.pdf
8. https://www.reddit.com/r/programming/comments/17tfgl/ntru_is_an_asymmetric_publicprivate_key/
9. https://gitlab.com/cyber5k/valens/-/blob/master/setup.py?ref_type=heads
10. https://github.com/smarky7CD/PyNTRU/blob/master/PyNTRU/ntru_poly_ops.py
11. https://ideaexchange.uakron.edu/cgi/viewcontent.cgi?article=1880&context=honors_research_projects
12. https://math.berkeley.edu/~apaulin/AbstractAlgebra.pdf

To My Mentors:

Dr. Deepak Mishra

Dr. Mahesh Sreekumar Rajasree

Thank You

Any Questions??

Every challenge(bug) teaches you how to live(code)🙏