Enhancing the NTRU Cryptosystem

Awnon Bhowmik
Department of Mathematics and Computer Science
CUNY York College
94 – 20 Guy R. Brewer Blvd
Jamaica, NY 11451

Unnikrishnan Menon
Department of Electrical and Electronics Engineering
Vellore Institute of Technology, Vellore
Tamil Nadu 632014

ABSTRACT

NTRU is an open-source public key cryptosystem that uses lattice-based cryptography to encrypt and decrypt data. Unlike other popular public-key cryptosystems, it is resistant to attacks using Shor's Algorithm and its performance has been shown to be significantly greater. This paper talks about how Koblitz encoding from Elliptic Curve Cryptography (ECC) can be used to convert each character in a dataset to a point on an elliptic curve. A sum of squares analogy is pitted against the cantor pairing function to turn the point to a single number, which is converted to a sequence of coefficients in $\mathbb{Z}$. A polynomial is then generated for each of these characters. Then the polynomial is reduced, and then shown that choosing appropriate parameters for the cryptosystem can make it highly secure and that the decryption algorithm turns out taking linear time. Since each character is represented by its own polynomial, it increases obscurity thereby increasing the complexity for decryption and thus the security level. A form of data compression has also been implemented and it has been tested whether data compression and expansion during the encryption-decryption process results in original data with no or minimal loss.

General Terms

Data Security

Keywords

Post quantum cryptography, lattice-based encryption, quantum cryptography, Koblitz encoding, post quantum cryptosystem, ntru cryptography, ntru cryptosystem.

1. INTRODUCTION

1.1 Why Lattice Cryptography

The real reason (Zhang, 2017)

- In 1994, Shor's Algorithm broke RSA and ECC with quantum computers
- 2015, NSA announcement: prepare for the quantum apocalypse
- 2017, NIST call for competition/standardization

Further usefulness (Zhang, 2017)

- Good understanding of underlying hard problem
- Fast, parallelable, hardware friendly
- Numerous applications, FHE, ABE, MMap, Obfuscation

Data vaulting attack (Zhang, 2017)

- Also known as harvest-then-decrypt attack
- Data needs to be secret for, let's say, 30 years
- Quantum computer arrives in, let's say, 15 years
- Perhaps the most practical attack in cryptography

Encrypt Schemes (Zhang, 2017)

- NTRUEncrypt – standardized by IEEE and ASC X9

Signature Schemes (Zhang, 2017)

- BLISS (NTRU)
- pqNTRUSign (NTRU)

This paper focuses on the NTRUEncrypt and NTRUDecrypt algorithm

2. KOBLITZ ENCODING AND DECODING ALGORITHM

The encoding algorithm (Brady, Davis, & Tracy, 2010) is as follows

- Given a message M, convert each character m_k into a number a_k using Unicode, where $b = 2^{16}$ and $0 < a_k < 2^{16}$
- Convert the message M into an integer using
$$m = \sum_{k=1}^{n} a_k b^{k-1}$$
In practice an $n \leq 160$ is chosen such that m satisfies
$$m \leq 2^{16\cdot 160} < p$$
- A number d is fixed such that $d \leq \frac{p}{m}$. In practice the prime p is chosen large enough so that $d = 100$ can be allowed.
- For integers $j = 0,1,2,\ldots d-1$, the following are performed
- x coordinate of a point on the elliptic curve is computed as
$$x_j = (dm + j)\ mod\ p \text{ where } m = \left\lfloor \frac{x_j}{d} \right\rfloor$$
- Compute $s_j = \left(x_j^3 + Ax + B\right) mod\ p$
- If $\left(s_j\right)^{\frac{p+1}{2}} \equiv s_j\ mod\ p$, then y coordinate of a point on the elliptic curve is defined as $y_j = \left(s_j\right)^{\frac{p+1}{4}}\ mod\ p$. Return the point $\left(x_j, y_j\right)$.

Thus, the message M is encoded as an element of the Abelian group $G = E\left(\mathbb{F}_p\right)$. The following is performed for decryption.

- Considering each point (x, y) and setting
$$m = \left\lfloor \frac{x-1}{k} \right\rfloor$$

Which is essentially means $a_k = \left\lfloor \frac{m}{b^{k-1}} \right\rfloor \; mod \; b$. Thus, each character is recovered and concatenated to produce the original message M.

3. EQUIVALENCE CLASS MODULO 3

Any number divided by 3 can result in remainders {0,1,2}. So, any number divided by 3 is one of $3r, 3r+1, 3r+2$. This is useful in constructing the trinary basis $\{-1,0,1\}^{\dim V}$. The elements in the basis forms the coefficients of the polynomial in (1).

Theorem 1. Let $R \subseteq S \times S$ be an equivalence class on a set S. Then the set of $\mathcal{R}$-classes constitutes the whole of S.

Proof.

$\forall x \in S: x \in [x]_{\mathcal{R}}$	Definition of equivalence class
$\neg(\exists x \in S: x \notin [x]_{\mathcal{R}})$	Assertion of universality
$\neg(\exists x \in S: x \notin \cup\, [x]_{\mathcal{R}})$	Definition of set union
$\forall x \in S: x \in \cup\, S/\mathcal{R}$	Assertion of universality
$S \subseteq \cup\, S/\mathcal{R}$	Definition of a subset

Also:

$\forall X \in S/\mathcal{R}: X \subseteq S$	Definition of equivalence class
$\cup\, S/\mathcal{R} \subseteq S$	Assertion of universality

By definition of set equality

$$\cup\, S/\mathcal{R} \subseteq S$$

And so, the set of all $\mathcal{R}$-classes constitutes the whole of S (Union of Equivalence Classes is Whole Set, n.d.)

4. CANTOR PAIRING FUNCTION

This is an elegant function proposed by the Russian mathematician George Cantor that takes in two natural numbers and turns it into a single number. This function is a primitive recursive pairing function. (Szudzik, 2006)

$$\pi: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

And is defined by

$$\pi(x,y) = \frac{1}{2}(x+y)(x+y+1) + y \qquad (2)$$

Due to the way its defined, this is a one-to-one and onto function, which means it is invertible. This consequently means that given a single number, it can be readily mapped back to a unique (x,y) ordered pair.

In order to retrieve an ordered pair (x,y) from a given t, the following transformations are used

$$\begin{aligned} \omega &= x+y \\ t &= \frac{1}{2}\omega(\omega+1) \\ z &= t+y \end{aligned}$$

From the second equation, cross multiplying gives a quadratic in ω

$$\omega^2 + \omega - 2t = 0$$

Solving it gives us

$$\omega = \frac{\sqrt{8t+1}-1}{2}$$

which is a strictly increasing and continuous function when t is non-negative real. Since

$$t \le z = t + y < t + (\omega+1) = \frac{(\omega+1)^2 + (\omega+1)}{2}$$

This implies that

$$\omega \le \frac{\sqrt{8z+1}-1}{2} < \omega + 1$$

And thus

$$\omega = \left\lfloor \frac{\sqrt{8z+1}-1}{2} \right\rfloor$$

Finally calculate x and y from z as follows

$$\begin{aligned} \omega &= \left\lfloor \frac{\sqrt{8z+1}-1}{2} \right\rfloor \\ t &= \frac{\omega^2+\omega}{2} \\ y &= z - t \\ x &= \omega - y \end{aligned}$$

5. LATTICE CRYPTOGRAPHY

What is a lattice? (Zhang, 2017)

A **lattice** is a (maximal) discrete subgroup of $\mathbb{R}^n$ for some $\mathbb{R}$-basis, or equivalently it consists of all the integral combinations of $d \le n$ linearly independent vectors over $\mathbb{R}$.

$$\mathcal{L} = \mathbb{Z}\boldsymbol{b_1} + \cdots + \mathbb{Z}\,\boldsymbol{b_d} = \{\lambda_1 \boldsymbol{b_1} + \lambda_2 \boldsymbol{b_2} + \cdots + \lambda_n \boldsymbol{b_n}:\ \lambda_i \in \mathbb{Z}\}$$

Where d is the dimension and $\boldsymbol{B} = (\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_n})$ is the $\mathbb{R}$ basis. Here is an example.

$$\boldsymbol{B} = \begin{pmatrix} 5 & \frac{1}{2} & \sqrt{3} \\ \frac{3}{5} & \sqrt{2} & 1 \end{pmatrix}$$

Here $d = 2. n = 3$ and hence $d \le n$.

The **discriminant** of $\mathcal{L}$ is the volume of a fundamental domain

$$Disc(\mathcal{L}) = \text{Vol}\{t_1\boldsymbol{b_1} + t_2\boldsymbol{b_2} + \cdots + t_n\boldsymbol{b_n}: 0 \le t_i < 1\}$$

Lattices have been extensively studied since (at least) the 19th century and have applications throughout mathematics, physics and computer science. For many applications, both theoretical and practical, one is interested in finding short non-zero vectors in $\mathcal{L}$. (Silverman, 2015)

Short Vectors – Theory (Silverman, 2015) A famous theorem of Hermite (1870s) says that a lattice $\mathcal{L}$ contains a non-zero vector $\boldsymbol{b} \in \mathcal{L}$ satisfying

$$||\boldsymbol{b}|| \le \gamma_n Disc(\mathcal{L})^{1/n}$$

The optimal value for γ_n, called **Hermite's constant**, is known only for $n \le 8$, but for large n the following holds true

$$\sqrt{n/2\pi e} \lesssim \gamma_n \lesssim \sqrt{n/\pi e}$$

The **shortest vector problem (SVP)** (Micciancio, 2005) is that of determining the shortest non-zero vector in $\mathcal{L}$. Hermite's theorem suggests that in a "random" lattice,

$$\min\{||\boldsymbol{b}||: \boldsymbol{0} \ne \boldsymbol{b} \in \mathcal{L}\} \asymp \sqrt{n} \cdot Disc(\mathcal{L})^{1/n}$$

The **closest vector problem (CVP)** (Micciancio, Closest Vector Problem, 2005) is that of determining the vector in $\mathcal{L}$ that is closest to a given non-lattice vector $\boldsymbol{\omega}$.

In low dimension it is not too hard to find short(est) vectors. But as the dimension increases, it becomes very hard. A computational breakthrough is the

LLL Algorithm 1982. Let $n = \dim(\mathcal{L})$ and let $r(\mathcal{L})$ denote the length of the shortest non-zero vector in $\mathcal{L}$. Then there is a polynomial time algorithm to find a non-zero vector $\boldsymbol{b} \in \mathcal{L}$ satisfying

$$||\boldsymbol{v}|| \leq 2^{\frac{n}{2}} \cdot r(\mathcal{L})$$

Many improvements have been made, but there is currently no algorithm that finds a vector satisfying

$$0 \neq ||\boldsymbol{b}|| \leq Poly(n) \cdot r(\mathcal{L})$$

faster than $O(1)^n$. This suggests using SVP and CVP as the basis for cryptographic algorithms.

- All crypto talks begin with an image of a dim-2 lattice
- There is an infinitude of bases which is formed due to matrix multiplication of $\boldsymbol{B}$ and some other matrix
- It involves solving the shortest vector problem (SVP). The exact version of the problem is only known to be NP-hard for randomized reductions (Ajtai, 1996). In this paper, something similar has been discussed in the proposed algorithm. Here a parameter t is calculated with the sum of squares and cantor pairing function separately, rather than the L^2 norm.

6. NTRUEncrypt

NTRU operations are based on objects in a truncated polynomial ring $R = \mathbb{Z}[X]/(X^N - 1)$ with convolution multiplication and all polynomials in the ring have integer coefficients and degree at most $N - 1$ (Stallings, 2017):

$$a(X) = a_0 + a_1X + a_2X^2 + \cdots + a_{N-1}X^{N-1} \qquad (1)$$

where $\boldsymbol{a} = (a_0, a_1, \cdots, a_{N-1})$. The product is denoted by $*$. In terms of convolutions, (Silverman, 2015)

$$\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b} \;\; with \;\; c_k = \sum_{i+j \equiv k \ (mod\ N)} a_i b_j$$

7. PROPOSED ALGORITHM

1. Input an arbitrary string containing alphanumeric and/or special characters.
2. Split into characters and pass through *Koblitz encoder*
3. Each character in the string is turned into a point on an elliptic curve $y^2 = x^3 + ax + b$, whose parameters a, b are user defined.
4. Convert point(s) (x, y) to a single number using the parametrization $t = x^2 + y^2$ or by using $t = \pi(x, y)$ and apply modulo 3 to turn it into ternary base number.

   $$\{0,1,2\} = \{0,1,-1\}$$

   Store these ternary numbers into a list. There should be multiple such lists due to multiple characters in the original string.
5. Find the maximum size among these lists. For the smaller lists, padding is applied to maintain consistency in the lists' sizes.
6. Using these ternary values from each list, every character in the string is represented as a polynomial.
7. Generate public and private keys. (NTRU, n.d.)
8. Public key is used to run NTRUEncrypt() and generate cipher text.
9. Private key is used to run NTRUDecrypt() and obtain original lists of ternary numbers.
10. Reapply padding.
11. A method called ternary_to_decimal() is applied to convert the padded lists into (x, y) points.
12. These points are passed through *Koblitz decoder* to obtain the list of characters. Combining these characters, finally facilitates the retrieval of the original plain text.

8. ALGORITHM FLOW DIAGRAM

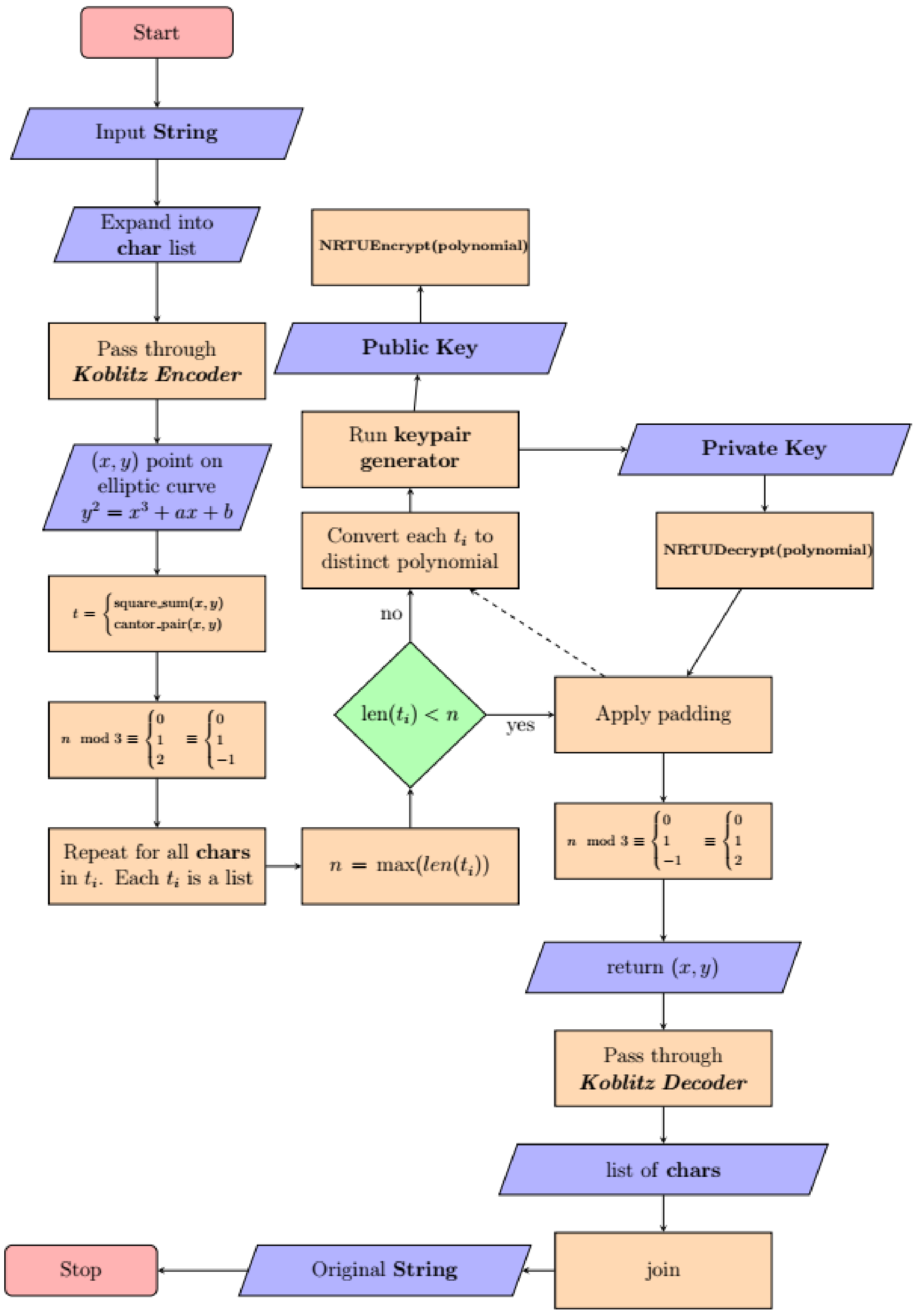

Start
Input String
Expand into char list
Pass through *Koblitz Encoder*
(x, y) point on elliptic curve $y^2 = x^3 + ax + b$
$t = \begin{cases} \text{square\_sum}(x,y) \\ \text{cantor\_pair}(x,y) \end{cases}$
$n \mod 3 \equiv \begin{cases} 0 \\ 1 \\ 2 \end{cases} \equiv \begin{cases} 0 \\ 1 \\ -1 \end{cases}$
Repeat for all chars in t_i. Each t_i is a list
$n = \max(len(t_i))$
$\text{len}(t_i) < n$
no
yes
Convert each t_i to distinct polynomial
Run keypair generator
Public Key
NRTUEncrypt(polynomial)
Private Key
NRTUDecrypt(polynomial)
Apply padding
$n \mod 3 \equiv \begin{cases} 0 \\ 1 \\ -1 \end{cases} \equiv \begin{cases} 0 \\ 1 \\ 2 \end{cases}$
return (x, y)
Pass through *Koblitz Decoder*
list of chars
join
Original String
Stop

Figure 1: Flow diagram for proposed algorithm

9. AN ATTEMPT ON LOSS LESS ENCODING

Messages encrypted by the Koblitz method results in a long list of polynomials. It is safe to assume that the length of this list increases linearly with the length of message. Hence, a modification to the algorithm in Figure 1. was attempted by converting the polynomials into a gray scale map to obtain some sort of data compression. Let's say Alice encrypts a message "Enhancing the NTRU cryptosystem". This algorithm will generate a ciphertext which is a collection of polynomials having the same order. It was observed that a convenient way to send this collection of polynomials would be to first map all the coefficients of the polynomials into the range of [0,1]. Then these values are used to make a greyscale mapping image. Alice sends over this image to Bob. On the other side, Bob takes the image and reads all the pixel values and can recover back the coefficients of the collection of polynomials using the secret key. Then he can continue with the decryption process.

Several attempts were tried to save the [0,1] pixel values into an image and read back the coefficients.

The first attempt involved PNG image format and produced results that were correct up to 2 decimal places only. This means that this format produced a large percentage error. Another thing to notice is that this involved only 8-bit precision unsigned integers. However, using TIFF image format with 64-bit precision unsigned integers resulted in a much better match between the encrypted and the decrypted matrix. But even then, there were a lot of loss during compression. So, this procedure may not be a feasible step to include in a revised algorithm. Following is a sample gray scale map that was generated for the message, “Enhancing the NTRU cryptosystem”.

Figure 2: Grayscale map of data compression

Remark: It is to be noted that every character in the input string is treated differently, i.e. if the message is “banana”, then each of the characters have a different random polynomial. The polynomials for all 3 a's and both n's are distinct. This is what makes this cryptosystem more secure.

Following is a sample test run of the lossless encoding program.

```
Enter Message: Awnon

Curve Parameters

Enter A: 9

Enter B: 7

Public Key = (-52x^14) + (38x^13) + (-1x^12) + (-25x^11) +
(10x^10) + (8x^9) + (-20x^8) + (39x^7) + (-32x^6) + (34x^5)
+ (-64x^4) + (-51x^3) + (10x^2) + (52x) + (29)

Encrypted =

[[62, 2, 14, 44, -16, 36, -35, -57, -62, -29, -47, 33, 6, 48, 25],
[-23, 4, -42, 54, -60, 23, -50, 17, 36, -9, 2, 41, 26, 34, -52],
[54, -51, -18, 23, -26, 2, -38, 32, 13, -19, 40, -15, 56, -43, -
36], [39, -57, 29, 22, -41, 43, 29, 8, -17, 5, -61, 20, 53, 60, -
56], [-49, 62, 6, 2, -52, 39, -49, 6, -37, 46, -4, 1, 26, -38, -35]]
```

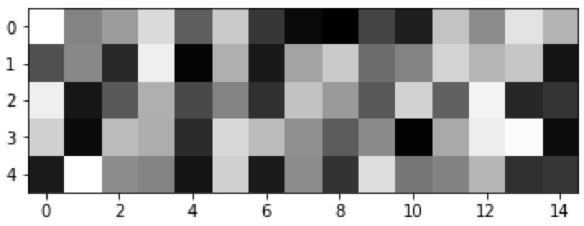


Figure 3: Grayscale map of sample test run

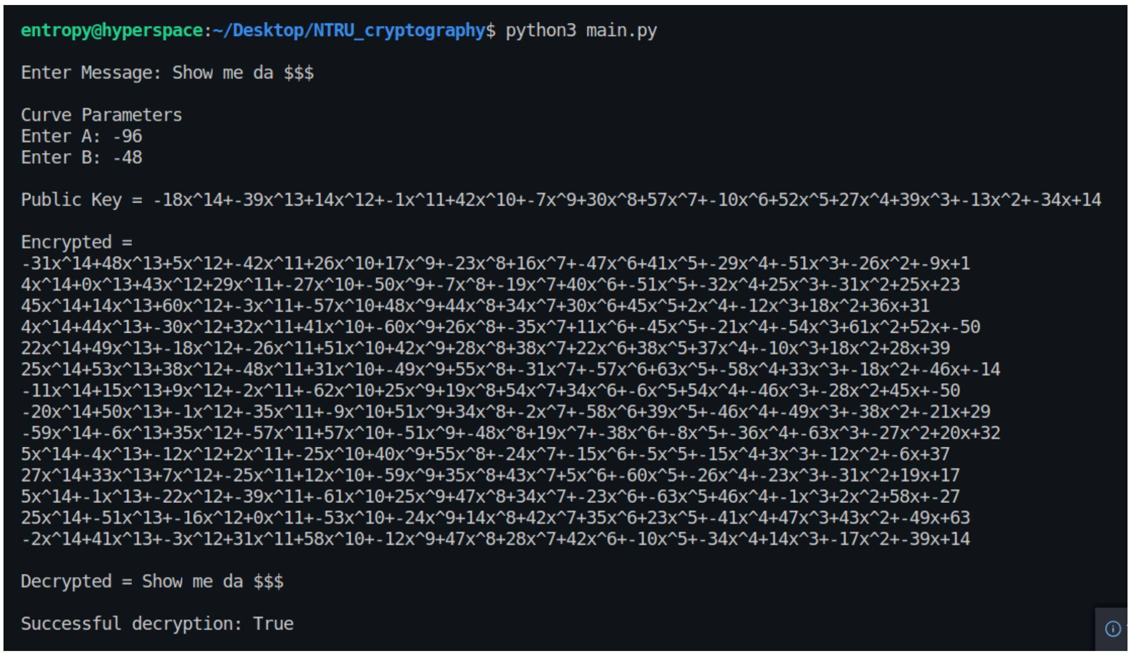
And here is the following attempt at decoding the coefficients of the polynomial from the grayscale image.

```
Data Type: float64
Min: 0.000, Max: 1.000
[[1. , 0.51612903, 0.61290323, 0.85483871, 0.37096774,
0.79032258,  0.21774194, 0.04032258, 0., 0.26612903,
0.12096774, 0.76612903, 0.5483871,  0.88709677,
0.7016129], [0.31451613, 0.53225806, 0.16129032,
0.93548387, 0.01612903, 0.68548387, 0.09677419,
0.63709677, 0.79032258, 0.42741935, 0.51612903,
0.83064516, 0.70967742, 0.77419355, 0.08064516],
[0.93548387, 0.08870968, 0.35483871, 0.68548387,
0.29032258, 0.51612903, 0.19354839, 0.75806452,
0.60483871, 0.34677419, 0.82258065, 0.37903226,
0.9516129, 0.15322581, 0.20967742], [0.81451613,
0.04032258, 0.73387097, 0.67741935, 0.16935484,
0.84677419, 0.73387097, 0.56451613, 0.36290323,
0.54032258, 0.00806452, 0.66129032, 0.92741935,
0.98387097, 0.0483871 ], [0.10483871, 1., 0.5483871,
0.51612903, 0.08064516, 0.81451613, 0.10483871,
0.5483871, 0.2016129,  0.87096774, 0.46774194,
0.50806452, 0.70967742, 0.19354839, 0.21774194]]
>>>print(decoded_coefficients)

[[62, 2, 14, 44, -16, 36, -35, -57, -62, -29, -47, 33, 6, 48, 25],
[-23, 4, -42, 54, -60, 23, -50, 17, 36, -9, 2, 41, 26, 34, -52],
[54, -51, -18, 23, -26, 2, -38, 32, 13, -19, 40, -15, 56, -43, -
36], [39, -57, 29, 22, -41, 43, 29, 8, -17, 5, -61, 20, 53, 60, -
56], [-49, 62, 6, 2, -52, 39, -49, 6, -37, 46, -4, 1, 26, -38, -35]]

>>>print(coefficients)

[[62, 2, 14, 44, -16, 36, -35, -57, -62, -29, -47, 33, 6, 48, 25],
[-23, 4, -42, 54, -60, 23, -50, 17, 36, -9, 2, 41, 26, 34, -52],
[54, -51, -18, 23, -26, 2, -38, 32, 13, -19, 40, -15, 56, -43, -
36], [39, -57, 29, 22, -41, 43, 29, 8, -17, 5, -61, 20, 53, 60, -
56], [-49, 62, 6, 2, -52, 39, -49, 6, -37, 46, -4, 1, 26, -38, -35]]
```

And finally, the following code snippet checks for matching elements in the two lists, containing encoded and decoded values.

```
>>>if coefficients == decoded_coefficients:
        print('Complete Match!')
    else:
        print('Not Equal!')
>>>Complete Match!
```

10. RESULT
The following is a snapshot of the console application program in action.



```
entropy@hyperspace:~/Desktop/NTRU_cryptography$ python3 main.py

Enter Message: Show me da $$$

Curve Parameters
Enter A: -96
Enter B: -48

Public Key = -18x^14+-39x^13+14x^12+-1x^11+42x^10+-7x^9+30x^8+57x^7+-10x^6+52x^5+27x^4+39x^3+-13x^2+-34x+14

Encrypted =
-31x^14+48x^13+5x^12+-42x^11+26x^10+17x^9+-23x^8+16x^7+-47x^6+41x^5+-29x^4+-51x^3+-26x^2+-9x+1
4x^14+0x^13+43x^12+29x^11+-27x^10+-50x^9+-7x^8+-19x^7+40x^6+-51x^5+-32x^4+25x^3+-31x^2+25x+23
45x^14+14x^13+60x^12+-3x^11+-57x^10+48x^9+44x^8+34x^7+30x^6+45x^5+2x^4+-12x^3+18x^2+36x+31
4x^14+44x^13+-30x^12+32x^11+41x^10+-60x^9+26x^8+-35x^7+11x^6+-45x^5+-21x^4+-54x^3+61x^2+52x+-50
22x^14+49x^13+-18x^12+-26x^11+51x^10+42x^9+28x^8+38x^7+22x^6+38x^5+37x^4+-10x^3+18x^2+28x+39
25x^14+53x^13+38x^12+-48x^11+31x^10+-49x^9+55x^8+-31x^7+-57x^6+63x^5+-58x^4+33x^3+-18x^2+-46x+-14
-11x^14+15x^13+9x^12+-2x^11+-62x^10+25x^9+19x^8+54x^7+34x^6+-6x^5+54x^4+-46x^3+-28x^2+45x+-50
-20x^14+50x^13+-1x^12+-35x^11+-9x^10+51x^9+34x^8+-2x^7+-58x^6+39x^5+-46x^4+-49x^3+-38x^2+-21x+29
-59x^14+-6x^13+35x^12+-57x^11+57x^10+-51x^9+-48x^8+19x^7+-38x^6+-8x^5+-36x^4+-63x^3+-27x^2+20x+32
5x^14+-4x^13+-12x^12+2x^11+-25x^10+40x^9+55x^8+-24x^7+-15x^6+-5x^5+-15x^4+3x^3+-12x^2+-6x+37
27x^14+33x^13+7x^12+-25x^11+12x^10+-59x^9+35x^8+43x^7+5x^6+-60x^5+-26x^4+-23x^3+-31x^2+19x+17
5x^14+-1x^13+-22x^12+-39x^11+-61x^10+25x^9+47x^8+34x^7+-23x^6+-63x^5+46x^4+-1x^3+2x^2+58x+-27
25x^14+-51x^13+-16x^12+0x^11+-53x^10+-24x^9+14x^8+42x^7+35x^6+23x^5+-41x^4+47x^3+43x^2+-49x+63
-2x^14+41x^13+-3x^12+31x^11+58x^10+-12x^9+47x^8+28x^7+42x^6+-10x^5+-34x^4+14x^3+-17x^2+-39x+14

Decrypted = Show me da $$$

Successful decryption: True
```

Figure 4: Sample test run of the entire process

It starts off by choosing the elliptic curve parameters. This generates a long polynomial as a public key. The orders of the polynomials are all equal to the order of the polynomial in the public key. Since every character gets its own n^{th} order polynomial, the encrypted version looks messy. But the decrypted string turns out to be an exact match with the original string showing that the method indeed works out fine.

11. EXPERIMENTS BY VARYING PARAMETERS

11.1 Valid elliptic parameters vs string length

1. A custom string of length 74 is taken such that it consists of all alphanumeric characters, special characters and space.
2. A $O(n^2)$ complexity loop is run to check for (a, b) in range $[-100,100]$.
3. Due to lack of recommended processing power, the program breaks out of the loop as soon as the first combination is found. The combination turned out to be $(a, b) = (-96, -48)$
4. The original string is shuffled 100 times to make 100 different strings of length 74.
5. For each of the shuffled strings, only the first 20 characters are considered due to shortage of processing power. So basically, then there will be 100 strings of length 20 that comprises of any random character on the English keyboard.
6. The encrypt-decrypt function is executed on these strings. The job of the encrypt-decrypt function is to return true only if the decrypted message matches perfectly with the input plaintext.

It was observed that for each of the 100 test cases, the encrypt-decrypt function returns true.

11.2 String length vs time to encrypt/decrypt
A function was written on Jupyter Notebook to test out the relationship between string length s (x-axis) and time of encryption/decryption t_{enc}/t_{dec} (y-axis). Following graph was obtained.

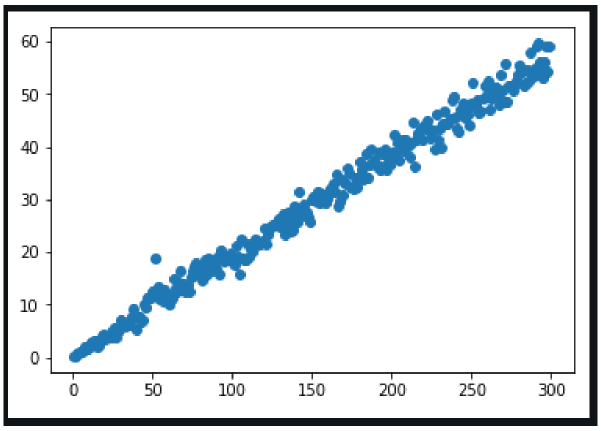


Figure 5: String length vs time to encrypt

A best fit shape through these points is a straight line through the *origin*. It can be safely assumed that for an arbitrary string length s, time to encrypt t_{enc} and time to decrypt t_{dec}, the following holds true

- $t_{enc} \propto s$
- $t_{dec} \propto s$

To test the authenticity of the encryption paradigm the elliptic curve parameters were fixed at $(a, b) = (-96, -48)$. A custom string of length 74 was generated that contained numeric values, alphabets (upper and lower case), special characters, spaces etc. and generated 100 new strings by randomly shuffling the elements of this custom string. It was observed that for each of the 100 test cases, the program was able to successfully retrieve the original message after completing an encrypt-decrypt cycle.

11.3 String length vs time to encrypt/decrypt

Here, the curve parameters were kept as $(a, b) = (-96, -48)$. A string of length 20 was considered that contained alphanumeric and special characters from the keyboard, with repetition. 100 such strings were generated by randomly shuffling all characters in the 20-character string. The following is a plot of test runs involving randomly shuffled strings vs the time of encrypt-decrypt cycle.

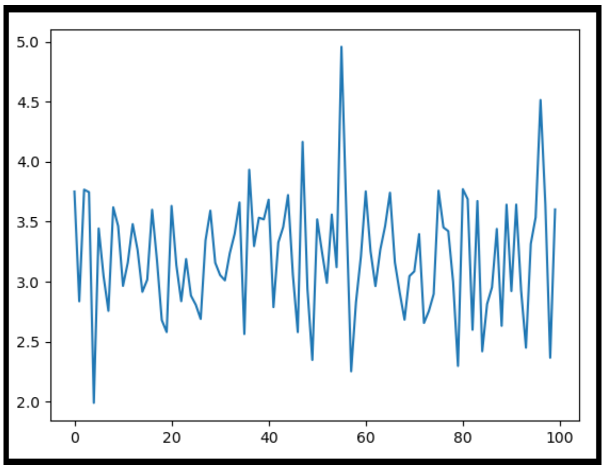


Figure 6: Shuffled String vs Encrypt-Decrypt Time

11.4 Comparison between sum of squares and cantor pairing function

A performance analysis was carried out between the two different parametrizations of (x, y), namely the sum of square and the cantor pair function. A variable string length of 100-1000 characters were tested for the encrypt-decrypt cycle using both the sum of square and the Cantor pairing function, one at a time. Following results were obtained.

Table 1: Time analysis – sum of squares vs cantor pairing function

String Length	Sum square (seconds)	Cantor pair (seconds)	Ratio
100	14.768	0.313	47.182
200	33.215	0.460	72.207
300	50.062	0.744	67.288
400	66.053	0.734	89.990
500	83.220	1.118	74.436
600	96.327	1.255	76.755
700	116.220	1.415	82.134
800	126.006	1.770	71.189
900	156.052	1.900	82.133
1000	165.771	2.102	78.863

Here is a plot showing the comparison between the two methods. Clearly the Cantor function is more efficient and helps keep the already complex algorithm as simple as possible. It can be clearly seen that at certain times, the Cantor pairing function is 90 times faster than the sum of squares parametrization.

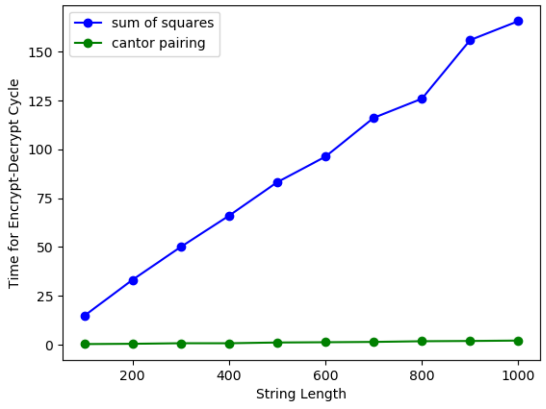


Figure 7: Effect of string length vs time of encrypt-decrypt cycle

12. CONJECTURE

1. The elliptic curve parameter $(a, b) = (-96, -48)$ works irrespective of the string length of the message and the type of characters present in the string.
2. $(a, b) = (-96, -48)$ is one of the many combinations of elliptic curve parameter that works. Proof of this is dependent on greater computational power.

13. FUTURE WORK

There are plans to wrap up the entire application into a pip install package which would be available for general public use. NTRU algorithm has been around for a while but there is still no official implementation of this algorithm in libraries such as pycryptodome. It would be nice if this program becomes a part of the existing library. It will serve as a great security layer against post quantum cryptography. Further, this application will be used as a security layer for a web socket based client server chat app, built with Tkinter. For the moment, the work done so far has been compiled into a GitHub repository (Menon & Bhowmik, 2020). It has been observed that using the NumPy module instead of the math module makes quick work of calculations that were coded from scratch. This would also fix some formatting issues and make the polynomial more legible.

14. CONCLUSION

This paper inspects a method that makes use of concepts stemming from the core of elliptic curve cryptography and the Cantor pairing function, in order to increase the complexity for decryption, thereby increasing the security level of the existing lattice based NTRU cryptosystem. An analysis has been performed on how introducing these concepts and tuning the appropriate set of parameters can lead to a much more stable and secure encryption – decryption cycle with a lower time complexity.

This research is significant because as of 2020, the most popular public-key algorithms can be broken by a sufficiently strong quantum computer. NTRU is resistant to attacks based on quantum computing, to which the standard RSA and ECC public-key cryptosystems are vulnerable to.

Since NTRU algorithm has been around for a while, it would be easier for hackers to find a way around it and funnel funds, liquid assets and/or confidential information from online accounts. The proposed method increases the complexity of breaching into such accounts by introducing additional layers of security that is purely based on fundamental mathematics while ensuring a practically feasible time for execution.

15. REFERENCES

[1] Ajtai, M. (1996). Generating Hard Instances of Lattice Problems. ACM Symposium on Theory of Computing. Philadelphia, Pennsylvania: Association for Computing Machinery. doi:https://dl.acm.org/doi/10.1145/237814.237838

[2] Brady, R., Davis, N., & Tracy, A. (2010, July). Encrypting with Elliptic Curve Cryptography. Retrieved from https://www.math.purdue.edu/~egoins/notes/Encrypting_Text_Messages_via_Elliptic_Curve_Cryptography.pdf.

[3] Menon, U., & Bhowmik, A. (2020, March). NTRU Cryptography. Retrieved from https://github.com/7enTropy7/NTRU_cryptography.

[4] Micciancio, D. (2005). Closest Vector Problem. SpringerLink. doi:https://doi.org/10.1007/0-387-23483-7_66.

[5] Micciancio, D. (2005). Shortest Vector Problem. SpringerLink. doi:https://doi.org/10.1007/0-387-23483-7_392.

[6] NTRU. (n.d.). Retrieved from LatticeHacks: https://latticehacks.cr.yp.to/ntru.html.

[7] Silverman, J. H. (2015, January 12-16). NTRU and Lattice-Based Crypto: Past, Present, and Future. The Mathematics of Post-Quantum Cryptography. Retrieved May 19, 2020, from http://archive.dimacs.rutgers.edu/Workshops/Post-Quantum/Slides/Silverman.pdf.

[8] Stallings, W. (2017). Finite Fields. In W. Stallings, The principles and Practice of Cryptography and Network Security 7th Edition (Vol. 20, pp. 123-152). Pearson Education.

[9] Szudzik, M. (2006). An Elegant Pairing Function. Washington, DC. Retrieved from http://szudzik.com/ElegantPairing.pdf.

[10] Union of Equivalence Classes is Whole Set. (n.d.). Retrieved from ProofWiki: https://proofwiki.org/wiki/Union_of_Equivalence_Classes_is_Whole_Set.

[11] Zhang, Z. (2017, July 12). A short review of the NTRU cryptosystem. Retrieved from https://www.slideshare.net/OnBoardSecurity/a-short-review-of-the-ntru-cryptosystem.