TOBB UNIVERSITY OF ECONOMICS AND THE TECHNOLOGY

FACULTY OF SCIENCE AND LETTERS

DEPARTMENT OF MATHEMATICS

NTRU-KE: A LATTICE BASED PUBLIC KEY EXCHANGE PROTOCOL

ESRA GÜNSAY and BAHADIR ÇOLAK

Supervisor: Doç. Dr. ÇETİN ÜRTİŞ

August 2017

CONTENTS

Pages

LIST OF ABBREVIATIONS

DH: Diffie-Hellman

DHP: Diffie-Hellman Problem

DL: Discreate Logarithm

NTRU: Nth Degree Truncated Polynomial Ring Units

NTL: Number Theory Library

CML: Convolution Modular Lattice

LIST OF FIGURES

ABSTRACT

A public cryptosystem primarily consists of three important members: 1) public key encryption, 2) digital signature, and 3) public key exchange

Public key exchange protocol is identified as an important application in the field of public-key cryptography. Most of the existing public key exchange schemes are Diffie-Hellman (DH)-type, whose security is based on DH problems over different groups. As we know that the public key exchange protocol in the NTRU cryptosystem is missing a NTRU lattice-based key exchange primitive in related to NTRU-ENCRYPT and NTRU-SIGN. We are going to explain theorical background of NTRU-KE and imlementation of NTRU key exchange on phyton.

1. INTRODUCTION

Key exchange is a trusted and functional method in cryptography. This cryptographic method centered exchanging keys between two parties, allowing use of a cryptographic algorithm.

To be able to exchanging ecrypted messages, both sides of process must be equipped to encrypt messages to be sent and decrypt messages received. The nature of the equipping they require depends on the encryption technique they might use. If they use a code, both will require a copy of the same codebook. If they use a cipher, they will need appropriate keys. If the cipher is a symmetric key cipher, both will need a copy of the same key. If an asymmetric key cipher with the public/private key property, both will need the other's public key.

Key Exchange Problem

The key exchange problem is based on findin a way to exchange whatever keys or other information needed which no one else can obtain a copy. Historically, this required trusted couriers, diplomatic bags, or some other secure channel.

With the advent of public key / private key cipher algorithms (i.e., asymmetric ciphers), the encrypting could be made public, since (at least for high quality algorithms) no one without the decrypting key could decrypt the message.

Roadmap

In cryptography security can be provided as using a hard problem. review the existing cryptosystems and the underlying hard problems from table 1.1 lattice based NTRU public key exchange protocal is missing. NTRU-KE is a non-DH-type scheme whose security is related to shortest vector problem (SVP) in a NTRU lattice.

hard problem / protocol	DL Problem	ECDL Problem	SVP and CVP in a NTRU lattice
public key encryption	ElGamal [19]	ECElGamal	NTRU-ENCRYPT [20]
digital signature	DSA [21]	ECDSA [22]	NTRU-SIGN [23, 24]
public key exchange	DH	ECDH	?

Figure 1-Representative Members in DL,ECDL, and NTRU Cryptosystems

The remainder of this paper is organized as follows. Section 2 gives some therical background of Diffie-Hellman, lattices and lattice problem. Section 3 provides some mathematical preliminaries about NTRU Cryptosystem. In Section 4, we describe NTRU-KE, formalize the underlying hard problem, and analyze the lattice attacks on NTRU and NTRU-KE. Finally, the implementation of NTRU-KE is in Section 5.

2. THEORETICAL BACKGROUND

2.1 Discreate Logarithm Problem

The term "discrete logarithm" is most commonly used in cryptography, although the term "generalized multiplicative order" is sometimes used as well [8]. In number theory, the term "index" is generally used instead [9]

If a is an arbitrary integer relatively prime to n and g is a primitive root of n, then there exists among the numbers 0, 1, 2, ..., $\Phi(\mathrm{n}) - 1$, where Φ(n) is the totient function, exactly one number μ such that

$$a = g^{\mu} \ (mod\ n)$$

The number μ is then called the discrete logarithm of a with respect to the base g modulo n and is denoted

$$\mu = ind_g a \ (mod\ n) \qquad [10]$$

There is no known efficient algorithm to solve this problem and it is intractable for large p of the order of, say 2^{1024} .

For example, the logarithm of 2 to the base 10 is approximately 0.30103; that is

$$log_{10}2 \approx 0.30103 \leftrightarrow 10^{0.30103} \approx 2.0$$

2.2 Diffie-Hellman Key Exchange

is a method of securely exchanging cryptographic keys over a public channel. Originally conceptualized by Ralph Merkle and named after Whitfield Diffie and Martin Hellman.[1] Diffie-Hellman key exchange was the first published public key

algorithm. [2] D–H is one of the earliest practical examples of public key exchange implemented within the field of cryptography.

2.2.1 Diffie-Hellman Problem

The motivation for this problem is that many security systems use mathematical operations that are fast to compute, but hard to reverse. For example, they enable encrypting a message, but reversing the encryption is difficult. If solving the DHP were easy, these systems would be easily broken.

2.2.2 Diffie-Hellman Algorithm

Diffie-Hellman is an algorithm used to establish a shared secret between two parties. It is primarily used as a method of exchanging cryptography keys for use in symmetric encryption algorithms like AES.

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secred communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

The purpose of the algorithm is to allow two users to send a key to each other in a secure manner, and then send the encrypted messages to each other with this key. Algorithm is limited by key exchange.

The Diffie-Hellman common secret key generation system is based on the discrete logarithm problem and its reliability is based on choosing very large prime numbers.[11],[12] p is a large prime number that it is not possible to solve the problem of discrete logarithm in Z_p .And let g, be a primitive root in Z_p . p and g known by

- A, $0 \leq a \leq p-2$ chooses a random number a which provides this inequality. Then computes $c = g^a \pmod p$ and send it to user B.
- B, $0 \leq b \leq p-2$ chooses a random number b which provides this inequality. Then computes $d = g^b \pmod p$ and send it to user A.
- A, computes commen secret key k as;

$$k = d^a = (g^b)^a$$

- B, computes commen secret key k as;

$$k = c^b = (g^a)^b$$

Thus, A and B agree on k, which is a common key between them.[13]

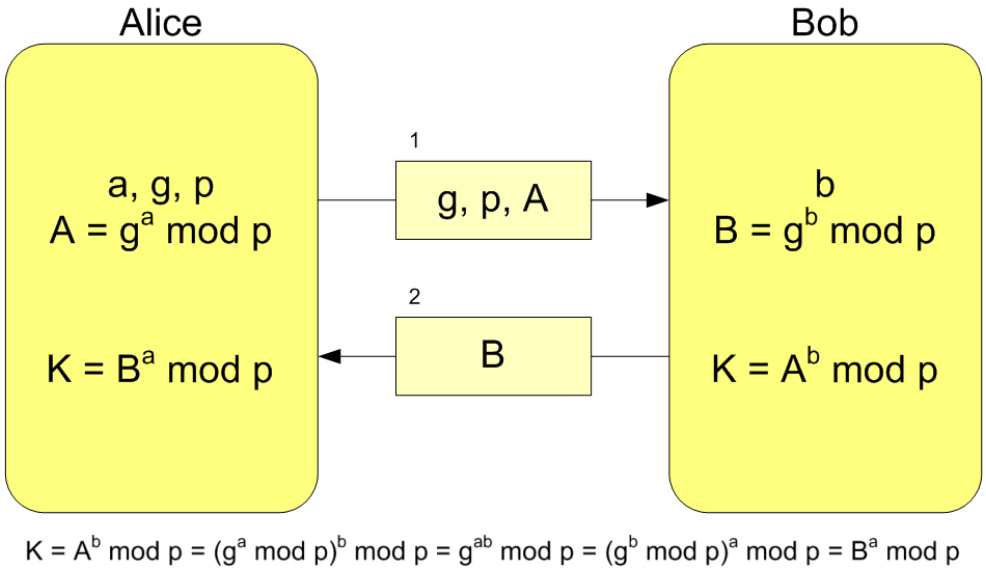

Figure 2-Diffie Hellman Key Exchange Scheme

Here is an example of the protocol, with non-secret values are p, g, A, B, and secret values are a and b.

Agree to use a modulus p = 541 and base g = 2.

A chooses its own private key a , a =137 and B chooses its own private key as b = 193 .

A computes to c;

$$c = g^a (mod\ p) \rightarrow 208 = 2137\ (mod\ 541)$$

Then sends it to the B. By the way B computes to d;

$$d = g^b\ (mod\ p) \rightarrow 195 = 2193\ (mod\ 541)$$

And sends it to the A. Then they compute the shared secret key k;

$$k = c^b = (g^a)^b\ (mod\ p) \rightarrow (2137)^{193}\ (mod\ 541)$$

$\rightarrow \quad (208)^{193} \ (mod\ 541)$

$\rightarrow \quad 486 \ (mod\ 541)$

2.2.3 Diffie-Hellman Cryptanalysis [3]

In the simple case of prime groups, Alice and Bob agree on a prime p and a generator g of a multiplicative subgroup modulo p. Alice sends g^a mod p, Bob sends g^b mod p, and each computes a shared secret g^{ab} mod p. While there is also a Diffie-Hellman exchange over elliptic curve groups, we address only the "mod p" case. The security of Diffie-Hellman is not known to be equivalent to the discrete log problem but computing discrete logs remains the best known cryptanalytic attack. An attacker who can find the discrete log x from $y = g^x$ mod p can easily find the shared secret. Textbook descriptions of discrete log can be misleading about the computational tradeoffs, for example by balancing parameters to minimize overall time to compute a single discrete log.

2.3 Integer Lattices [5]

If $a_1, a_2, \dots, a_n$ are n independent vectors in $\mathbb{R}^m$, $n \leq m$, then the integer lattice with these vectors as basis is the set $L = \{\sum_1^n x_i a_i : x_i \in \mathbb{Z}\}$. A lattice is often represented as matrix A whose rows are the basis vectors $a_1, a_2, \dots, a_n$. The elements of the lattice are simply the vectors of the form $v^T A$, which denotes the usual matrix multiplication. We will specialize for now to the situation when the rank of the lattice and the dimension are the same ($n = m$).

As an example Fig is a lattice ina Euclidian plane.

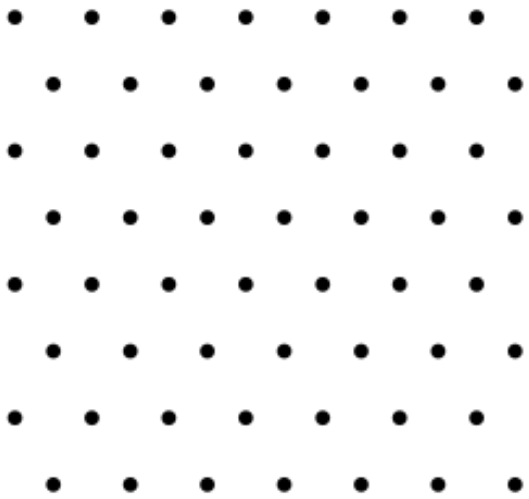

Figure 3 - A Lattice in the Euclidian Plane

And Fig shows five lattices in Euclidian plane.

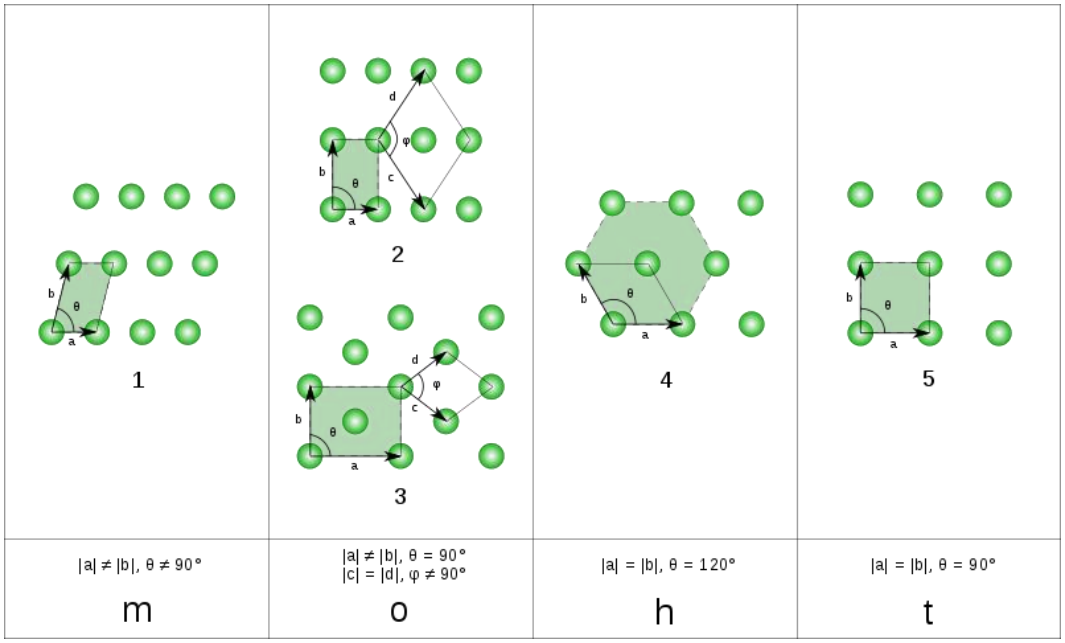

Figure 4 - Five lattices in the Euclidean plane

2.4 Lattice Problem

A class of optimization problems which takes place on lattices named Lattice Problem. Lattice problems hardness is central point construction of secure lattice based cryptosystems.

Set of all integer linear combinations of basis vectors $B = [b_1, \dots, b_n] \subset R^n$

$$L(B) = \{Bx : x \in \mathbb{Z}^n\} \subset \text{span (B)} = \{Bx : x \in R^n\}$$

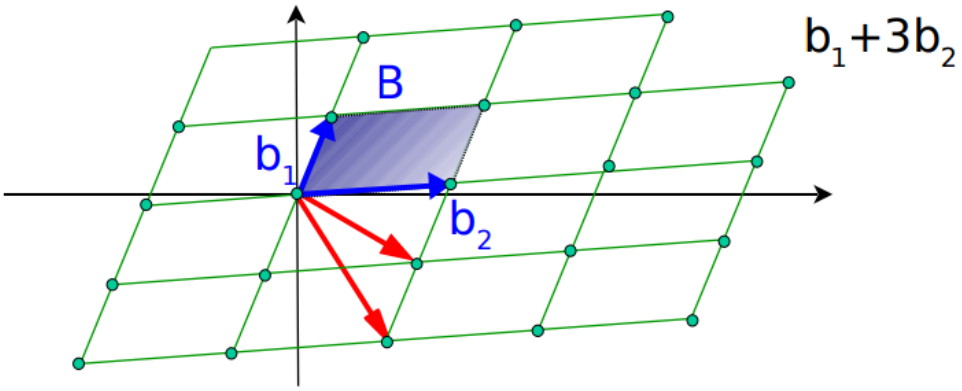

Figure 5- Point Lattices

[15]

The two fundamental problems in the theory of integer lattices are the shortest vector problem (SVP), and the more general closest vector problem (CVP).

Roughly speaking, both of these problems are concerned with finding the "most efficient" basis of the lattice - a basis consisting of vectors which are as short and as orthogonal as possible. Specifically, if $\boldsymbol{v}$ is a vector in the lattice, let $||v||$ denote a norm on v, typically the sup norm, $\max \boldsymbol{v_i}$, or the Euclidean norm $\sqrt{\sum v_i^2}$. Given a basis A for a lattice L, the shortest vector problem is that of finding a nonzero vector in L with minimum norm. Given an arbitrary (target) vector $\boldsymbol{v}$ in $\mathbb{Z}^n$, the CVP is the problem of finding the lattice point $x^T A$ closest in norm to v, i.e., of minimizing $||Ax - v||$. No polynomial time algorithms exist for solving either of these problems, although they have been well studied both experimentally and theoretically. There are algorithms which find approximations to the shortest vector - we'll say more on that in a moment.

Fig is an illustration of SVP (basis vectors in blue, shortest vector in red) and fig is an illustration of the closest vector problem (basis vectors in blue, external vector in green, closest vector in red).

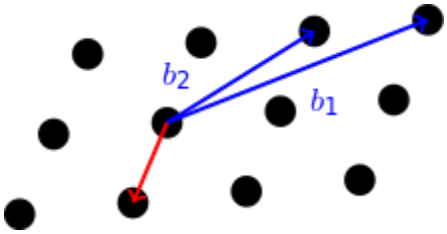

Figure 6- an illustration of the shortest vector problem

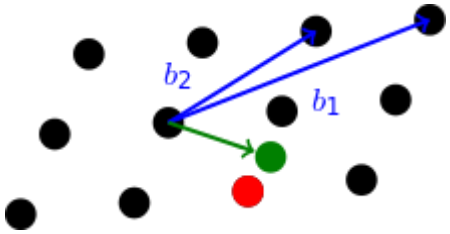

Figure 7- an illustration of the closest vector problem

3. NTRU CRYPTOSYSTEM

NTRU is the first public key cryptosystem not based on factorization or discrete logarithmic problems. NTRU is a lattice-based alternative to RSA and ECC and is based on the shortest vector problem in a lattice. [6]

3.1 BRIEF HISTORY

NTRU was founded in 1996 by 3 mathematicians: Jeffrey Hoffstein, Joseph H. Silverman, Jill Pipher Later (at the end of 1996), these 3 mathematicians + Daniel Lieman founded the NTRU Cryptosystems, Inc, Boston, USA. The mathematicians were considered on speeding up the process. In 2009, NTRU Cryptosystem has been approved for standardization by the Institute of Electrical and Electronics Engineers (IEEE) [4]

3.2 Polynomial Ring [5]

The basic NTRU operations take place in the ring of convolution polynomials $R=\mathbb{Z}[X]/(X^N-1)$, where N is prime. An element F ∈ R will be written as a polynomial or a vector,

$$F = \sum_{i=0}^{N-1} f_i x^i = [F_0, F_1 \dots F_{N-1}]$$

3.3 NTRU Encryption Algorithm

Public parameters

A choice of N determines the polynomial ring R=$\mathbb{Z}[X]$/(X^N-1) Two moduli p and q are selected so that gcd(p, q) = 1 - here q will typically be a power of 2, and p will be very small. One example is (N, p, q) = (251, 3, 128). Additional public parameters are numbers, which, for reasons that will be apparent momentarily, we'll denote d_f, d_g, d_m and d_r. These specify the space of allowable private keys f and g the allowable messages, and the form of the random polynomial r used in encryption.[5]

Key Generation

User B randomly chooses 2 small polynomials f and g in the R (the ring of truncated polynomials). The values of these polynomials should kept in a secret. A chosen polynomial must have an inverse.

The inverse of f modulo q and the inverse of f modulo p will be computed as ;

$$f * f_q^{-1} = 1 \ (mod\ q)$$

$$f * f_p^{-1} = 1 \ (mod\ p)$$

Product of polynomials will be computed;

$$h = f_q^{-1} g \ (mod\ q)$$

This h is the public key, while both f and g are private.

Here is an easy example of key generation ;

Public parameters $(N, p, q, d) = (7,29,491531,2)$.

User B randomly chooses 2 small polynomials f and g in the R .

$$f(x) = x^6 + x^4 + x^2 + x + 1$$

$$g(x) = -x^6 + x^4 + x^3 - 1$$

B computes inverces of $f(x)$;

$$F_{q(x)} = f(x)^{-1}\ (mod\ q)$$
$$= 318457\ x^6 + 470762x^5 + 76153x^4 + 48461x^3 + 477685x^2 + 214612x + 359995(mod 491531)$$

$$F_{p(x)} = f(x)^{-1}\ (mod\ p) = 15x^6 + 4x^5 + 24x^4 + 10x^3 + 22x^2 + 6x + 1\ (mod\ 29)$$

Here $F_{p(x)}$ is private key of B.

$$h(x) = p * F_q * \text{g (mod q)}$$
$$= 20x^6 + 40x^5 + 2x^4 + 38x^3 + 8x^2 + 26x + 30(mod 41)$$

the polynomial h is public key of B.

Encryption

Choose a random polynomial r in d_r Compute the ciphertext ;

$$e = m + pr * h\ \ (mod\ q)$$

An example of NTRU Encryption. A decides to send b the message m.

$$m = x^6 + x^4 + x^2 + x + 1$$

Using the ephemiral key r;

$$r(x) = -x^6 - x^5 + x^3 + x^2$$

Then as we know encrypted message computtes as;

$$e = m + pr * h\ \ (mod\ q)$$

$$e(x) = 31x^6 + 19x^5 + 4x^4 + 2x^3 + 40x^2 + 3x + 25(mod\ 491531)$$

Where $(N, p, q, d) = (7, 29, 491531, 2)$.

Decryption

Multiply e by the private key f to obtain ;

$$f * e = f * m + pr * f * h\ \ (mod\ q)\ =\ f * m + pr * q\ \ (mod\ q)$$

where this last equality uses the definition of h .

reducin each of the coefficients of a modulo p. The polynomial $b = a\ (mod\ p)$ computed.

Using other private polynomial F_p ,

$$c = f_p * b\ \ (mod\ p)$$

computed. Which is the original message m.

An example of NTRU Decryption;

First B computes $a = f * e\ \ (mod\ q)$;

$$a = 60x^6 + 491530x^5 + 491444x^4 + 491445x^3 + 61x + 58\ (mod\ 491531)$$

Secondly B centerlifts modulo q to obtain

$$b = a\ (mod\ p) = 2x^6 + 9x^5 + 10x^4 + 11x^3 + 3x\ (mod\ 29)$$

Lastly B reduces a(x) modulo p and computes

$$c = f_p(x) * b(x)\ (mod\ p)$$

Centerlifting modulo p retrieves A's plain text

$$m = x^6 + x^4 + x^2 + x + 1$$

Where $(N, p, q, d) = (7, 29, 491531, 2)$.

```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:\Users\bahdr\Desktop\ntru-master\example_bobalice.py ======
Bob Will Generate his Public Key using Parameters
N=7,p=29 and q=491531
f(x)=  [1, 1, -1, 0, -1, 1]
g(x)=  [-1, 0, 1, 1, 0, 0, -1]
d   =  2
Public Key Generated by Bob:  [394609, 27692, 62307, 263073, 346149, 41538, 339225]
--------------------------------------------------
Alice's Original Message    :  [1, 0, 1, 0, 1, 1, 1]
Alice's Random Polynomial   :  [-1, -1, 1, 1]
Encrypted Message           :  [283889, 269991, 484569, 353054, 179995, 159222, 235409]
--------------------------------------------------
Bob decrypts message sent to him
Decrypted Message           :  [1, 0, 1, 0, 1, 1, 1]
This took 0.10 seconds
>>>
```

3.4 The Advantages of NTRU

NTRU provides us more efficient encryption and decryption, in both hardware and software implementations. On the other hand much faster key generation allowing the use of “disposable" keys (because keys are computationally “cheap" to create).

Low memory use allows it to use in applications such as mobile devices and Smart-cards [6].

4. NTRU-KE CONSTRUCTION

Former studies have led to a strong perception that, a key exchange primitive protocol typically consists of two flows. Classic DH protocol and related DH protocols have supported perception.

On the other hand, in NTRU Encryption two-flow-based key exchange primitive can not be desined because the special structure of the public key in NTRU Encrypt. As [7], This is the main reason why the NTRU-KE protocol has been missing up till now.

Nevertheless a key exchange primitive may have more than two flows like many applicable key exchange protocols have. It does not have to be only two flows.

Therefore, by this reason in [7] the three-flow-based primitive protocol is proposed in the name of NTRU-KE.

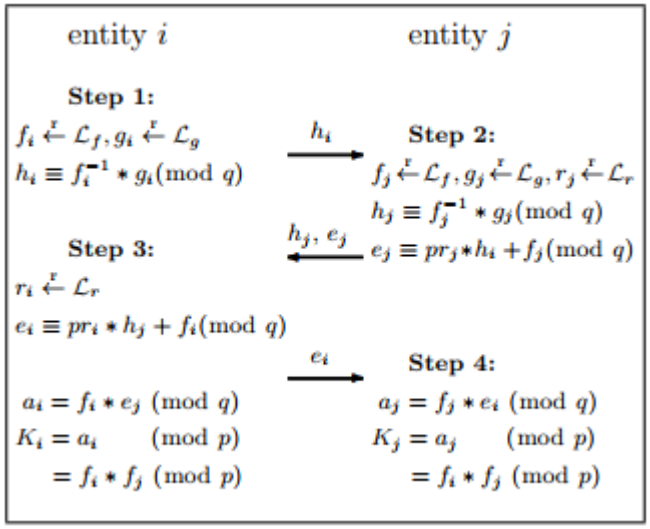

Figure 8-Data Flows in NTRU-KE

This Figure shows NTRU-KE data flows.

Firstly as NTRU Encryption, NTRU-KE protocol starts by determining 3 global integer (N, p, q).

- **Step 1:** entity i picks $f_i \rightarrow L_f$ such that there exists the inverse of f_i in R_q and picks $g_i \rightarrow L_g$. Then, he computes $h_i \equiv f_i^{-1} * g_i \pmod{q}$ and sends h_i to j .
- **Step 2:** upon receipt of h_i entity j picks $g_j \rightarrow L_g$, $r_j \rightarrow L_r$. Then, he computes $h_j \equiv f_j^{-1} * g_j \pmod{}$ and $e_j \equiv pr_j * h_i + f_j \pmod{q}$. Afterwards, he sends h_i and e_j to i .
- **Step 3:** upon receipt to h_j and e_i, entity j picks $r_j \rightarrow L_r$ and computes $e_i \equiv pr_i * h_j + f_i \pmod{q}$. Then he sends e_i to i and computes $a_i = f_i * e_j \pmod{q}$, $K_i = a_i \pmod{p} = f_i * f_j \pmod{p}$
- **Step 4:** upon receipt of e_i , entity j computes $a_j = f_j * e_i \pmod{q}$, $K_j = a_j \pmod{p} = f_i * f_j \pmod{p}$

It can be easily checked that the resultant common session key is

$$K_i = K_j = f_i * f_j \pmod{p} \qquad [7]$$

As we know DH type protocol consists of two flows, however NTRU-KE protocol consists three flow. Also in NTRU Encryption we need to compute inverse of f modulo p. In NTRU-KE it does not need to be computed. In Chapter 2 we refer DH type protocols adventages, NTRU-KE protocal features are as elegant and simple as DH-type protocols.

4.1 Lattice Attacks

The Shortest Lattice Vector Problem (SVP) consists in finding $x \in \mathbb{Z}^n \backslash 0$ minimizing $\| B \cdot \boldsymbol{x} \|$, where $B \in \mathbb{Q}^{\boldsymbol{mxn}}$ is given as input. The Closest Lattice Vector Problem (CVP) consists in finding $x \in \mathbb{Z}^n$ minimazing $\| B \cdot \boldsymbol{x} - \boldsymbol{t} \|$, where $B \in \mathbb{Q}^{\boldsymbol{mxn}}$ and $t \in \mathbb{Q}^{\boldsymbol{m}}$ are given as inputs.

4.1.1 Lattice Attacks on NTRU [16]

The best attack known against NTRU is based on lattice reduction, but this does not mean that lattice reduction is necessary to break NTRU. The simplest lattice-based

attack can be described as follows. Coppersmith and 166 Phong Q. Nguyen and Jacques Stern Shamir [17] noticed that the target vector $f \| g \in \mathbb{Z}^{2N}$ (the symbol $\|$ denotes vector concatenation) belongs to the following natural lattice:

$$L_{CS} = \{F \| G \in \mathbb{Z}^{2N} \mid F \equiv h * G \ mod\ q\ where\ F, G \in R\}$$

It is not difficult to see that L_{CS} is a full-dimensional lattice in $\mathbb{Z}^{2N}$, with volume q^N. The volume suggests that the target vector is a shortest vector of L_{CS} (but with small gap), so that a SVP-oracle should heuristically output the private keys f and g. However, based on numerous experiments with Shoup's NTL library [18], the authors of NTRU claimed in [19] that all such attacks are exponential in N, so that even reasonable choices of N ensure sufficient security. The parameter N must be prime, otherwise the lattice attacks can be improved due to the factorization of $X^N - 1$

NTRU without padding cannot be semantically secure since $e(1) \equiv m(1)\ (mod\ q)$ as polynomials, and it is easily malleable using multiplications by X of polynomials (circular shifts). And there exist simple chosen ciphertext attacks [20] that can recover the secret key.

4.1.2 Lattice Attacks on NTRU-KE

As we mentioned previously sections security of NTRU Encryption Sheme is related to CVP and SVP in a L_h NTRU lattice.

L_h is the set of vectors,

$$L_h = \{[f, g] \in \mathbb{Z}^{2N} \mid f * h \equiv g(mod\ q)\}$$

The best attack known against NTRU Encryption is based on lattice reduction. Likewise, in the light of this information, the best attack known against NTRU-KE must be these lattice attacks. Now two of them lattices attacks will be explained.

1) Lattice attack on h_i (or h_j). Suppose that a passive advesary A tries to recover secret information $[\mathbf{f_i}, \mathbf{g_i}]$ from $\mathbf{h_i}$. Subscript is insignificant.

By $\mathbf{h} \equiv \mathbf{f}^{-1} * \mathbf{g}$ (mod q), there exist $\boldsymbol{u} \in R_q$ with

$$\mathbf{u} = \frac{-\mathbf{f} * \mathbf{h} + \mathbf{g}}{\mathrm{q}}$$

Such that

$$[\mathbf{f}, \mathbf{u}].\mathrm{L_h} = [\mathbf{f}, \mathbf{g}]$$

Therefore, L_h contains the vector [**f , g**]. By choosing appropriate parameters, then the vector [**f , g**] is quite short, so it can be found by solving SVP (or approximate SVP) in L_h Indeed, this lattice attack corresponds to the lattice attack on a NTRU-ENCRYPT private key.

2) Lattice attack on e_i (or e_j) . Based on current knowledge of e_i ,

$$e_i \equiv pr_i * h_j + f_i$$

we can rewrite this relation in the vector form as,

$$[\mathbf{0}, \mathbf{e_i}] \equiv \left[\mathbf{r_i}, \mathbf{r_i} * (\mathrm{p}\mathbf{h_j}) (\mathrm{mod\ q})\right] + [-\mathrm{r_i}, f_i]$$

$\left[\mathbf{r_i}, \mathbf{r_i} * (\mathrm{p}\mathbf{h_j}) (\mathrm{mod\ q})\right]$ is in the lattice L_{ph_j}

The point [**0, $\mathbf{e_i}$**] is only $[-\mathrm{r_i}, f_i]$ away from a lattice point of L_{ph_j} . By choosing appropriate parameters, then the vector $[-\mathrm{r_i}, f_i]$ is quite short, so $[-\mathrm{r_i}, f_i]$can be recovered if we can find a vector inL_{ph_j} that is closest to the vector [**0, $\mathbf{e_i}$**]
This lattice attack corresponds to the lattice attack on a NTRU-ENCRYPT message.

It can be easily found that either of the above two lattice attacks, if succeed, will lead to a recovery of the session key. They indicate how NTRU-KE relate to SVP and CVP. The ideal lattice basis reduction algorithms can be used to solve SVP and CVP in NTRU lattice, and therefore, they can be applied to carry out the above two lattice attacks. However, the lattice attacks do not necessarily imply that NTRU-KE is insecure, as currently known lattice reduction algorithms still remain exponential time algorithms in terms of security parameters.

5. EXPERIMENTAL RESULTS

In this section we will give you a result of implementations of NTRU and NTRU-KE. For our experiment we built a python package[14] and run it on a computer with 8Gb RAM, Intel Core i7-6500U up to 3.16GHz. We use python for this implementation, because python is really efficient for mathematical calculations. To obtain the best results of encryption and decryption, we randomly chose polynomials ourselves, instead of using python libraries. A brief description of both packages is provided below.

NTRU.py: It take N,p and q parameters to do the initialization

1- genPublicKey(self,f,g,d): Takes f,g and d parameters and calculate f_p^{-1} and f_q^{-1} using Extended Euclid Algorithm. Then, generates H_i and returns it.
2- getPublicKey(self): Returns H_i
3- setPublicKey(self,public_key): Sets H_i to a given parameter.
4- encrypt(self,message,randPol): Produces an Encrypted message with a given random polynomial and the public key which is produced before.
5- decrypt(self,encryptedMessage): Decrypt a given message.
6- Helper Functions() :
    a. Checks if numbers are prime.
    b. Calculates the module.
    c. Prints all values.
    d. Checks if inputs satisfy conditions.

```
def genPublicKey(self,f_new,g_new,d_new):
        # Using Extended Euclidean Algorithm for Polynomials
        # to get s and t. Note that the gcd must be 1
        self.f=f_new
        self.g=g_new
        self.d=d_new
        [gcd_f,s_f,t_f]=poly.extEuclidPoly(self.f,self.D)
        self.f_p=poly.modPoly(s_f,self.p)
        self.f_q=poly.modPoly(s_f,self.q)
        self.h=self.reModulo(poly.multPoly(self.f_q,self.g),self.D,self.q)
        if not self.runTests():
                print "Failed!"
                quit()

def getPublicKey(self):
        return self.h

def setPublicKey(self,public_key):
        self.h=public_key
```

Poly.py(): Users can run this class with rational coefficient polynomials which are represented below;

$$c_1 = [1, -1, 1, 0, 1, -1] = x^5 - x^4 + x^3 + x - 1$$

1- resize(c1,c2): Adds leading zeros to the polynomial vectors.
2- trim(seq): Removes leading zeros.
3- subPoly(c1,c2): Subtracts two polynomials.
4- addPoly(c1,c2): Adds two polynomials.
5- multPoly(c1,c2): Multiplies two polynomials.
6- divPoly(c1,c2): Divides two polynomials.
7- modPoly(c,k): Calculates mod of each coefficient of c1.
8- cenPoly(c,q): Centerlift of Polynomial with respect to q.
9- extEuclidPoly(a,b): Extended Euclidean Algorithm.

```
# Subtracts two Polnomials
def subPoly(c1,c2):
        [c1,c2]=resize(c1,c2)
        c2=map(neg,c2)
        out=map(add, c1, c2)
        return trim(out)

# Adds two Polynomials
def addPoly(c1,c2):
        [c1,c2]=resize(c1,c2)
        out=map(add, c1, c2)
        return trim(out)

# Multiply two Polynomials
def multPoly(c1,c2):
        order=(len(c1)-1+len(c2)-1)
        out=[0]*(order+1)
        for i in range(0,len(c1)):
                for j in range(0,len(c2)):
                        out[j+i]=out[j+i]+c1[i]*c2[j]
        return trim(out)
```

Lets start with results of NTRU;

With parameters N = 251, p = 3 and q = 128

$f(x) = x^5 + x^4 - x^3 - x + 1$,

$g(x) = -x^6 + x^4 + x^3 - 1$

$m(x) = x^6 + x^4 + x^2 + x + 1$,

all the process(NTRU) took 2.07 sec:

Key Generation :

```
Public key genaretion took 1.76 seconds
Public Key Generated by Bob:  [15L, 96L, 97L, 16L, 63L, 34L, 29L, 52L, 24L, 127L
, 20L, 2L, 118L, 115L, 24L, 73L, 67L, 3L, 101L, 79L, 16L, 127L, 115L, 118L, 62L,
 39L, 11L, 31L, 52L, 84L, 68L, 36L, 53L, 15L, 22L, 89L, 78L, 101L, 112L, 56L, 45
L, 34L, 22L, 84L, 55L, 18L, 25L, 55L, 69L, 77L, 127L, 108L, 33L, 83L, 0L, 64L, 1
17L, 125L, 37L, 24L, 66L, 94L, 12L, 69L, 113L, 112L, 47L, 122L, 97L, 24L, 8L, 91
L, 20L, 126L, 6L, 75L, 116L, 65L, 59L, 75L, 25L, 127L, 20L, 123L, 103L, 122L, 2L
, 95L, 15L, 99L, 52L, 12L, 88L, 8L, 33L, 63L, 46L, 65L, 6L, 89L, 28L, 80L, 17L,
18L, 66L, 4L, 127L, 6L, 41L, 31L, 5L, 33L, 7L, 16L, 93L, 79L, 116L, 100L, 93L, 1
21L, 9L, 96L, 34L, 90L, 88L, 89L, 65L, 80L, 111L, 98L, 117L, 124L, 24L, 87L, 84L
, 10L, 102L, 99L, 0L, 25L, 67L, 83L, 13L, 95L, 88L, 23L, 123L, 110L, 6L, 39L, 67
L, 87L, 4L, 116L, 44L, 92L, 125L, 79L, 102L, 25L, 110L, 125L, 8L, 40L, 53L, 2L,
62L, 100L, 103L, 74L, 89L, 23L, 69L, 53L, 31L, 84L, 121L, 75L, 24L, 104L, 85L, 1
01L, 61L, 120L, 50L, 86L, 52L, 93L, 17L, 112L, 127L, 26L, 25L, 96L, 72L, 51L, 20
L, 102L, 22L, 59L, 60L, 81L, 27L, 91L, 65L, 47L, 92L, 19L, 47L, 82L, 10L, 127L,
39L, 123L, 100L, 12L, 0L, 96L, 9L, 127L, 126L, 97L, 70L, 17L, 52L, 64L, 89L, 50L
, 74L, 116L, 111L, 94L, 41L, 95L, 69L, 9L, 7L, 56L, 53L, 71L, 108L, 12L, 93L, 65
L, 65L, 32L, 114L, 18L, 96L, 17L, 33L]
```

Encryption :

```
Alice's Original Message    :  [1, 0, 1, 0, 1, 1, 1]
Alice's Random Polynomial   :  [-1, -1, 1, 1]
Encryption took 0.02 seconds
Encrypted Message           :  [68L, 73L, 78L, 122L, 87L, 49L, 49L, 48L, 89L, 46L
, 43L, 3L, 81L, 7L, 71L, 24L, 125L, 81L, 108L, 54L, 27L, 111L, 71L, 114L, 58L, 1
2L, 6L, 49L, 29L, 102L, 49L, 96L, 61L, 108L, 28L, 127L, 122L, 52L, 118L, 33L, 80
L, 11L, 7L, 47L, 7L, 99L, 32L, 107L, 13L, 58L, 16L, 117L, 61L, 101L, 46L, 28L, 9
0L, 106L, 57L, 31L, 88L, 87L, 80L, 109L, 28L, 80L, 69L, 40L, 76L, 16L, 49L, 66L,
 19L, 115L, 65L, 67L, 79L, 84L, 73L, 13L, 72L, 74L, 115L, 27L, 19L, 10L, 50L, 0L
, 42L, 77L, 5L, 22L, 25L, 32L, 49L, 0L, 52L, 83L, 114L, 48L, 118L, 89L, 60L, 91L
, 39L, 23L, 115L, 67L, 124L, 55L, 33L, 102L, 116L, 45L, 49L, 65L, 126L, 124L, 6L
, 6L, 61L, 71L, 0L, 71L, 112L, 97L, 72L, 96L, 17L, 64L, 56L, 32L, 73L, 6L, 59L,
51L, 49L, 63L, 39L, 16L, 21L, 9L, 116L, 126L, 123L, 119L, 111L, 18L, 90L, 52L, 3
0L, 57L, 45L, 102L, 49L, 80L, 85L, 52L, 108L, 103L, 10L, 60L, 6L, 49L, 120L, 107
L, 87L, 63L, 95L, 83L, 120L, 67L, 85L, 98L, 24L, 21L, 21L, 13L, 62L, 76L, 114L,
82L, 81L, 15L, 104L, 7L, 96L, 101L, 84L, 48L, 125L, 56L, 52L, 96L, 33L, 122L, 35
L, 3L, 97L, 123L, 15L, 76L, 33L, 69L, 112L, 18L, 51L, 3L, 91L, 74L, 50L, 104L, 3
4L, 53L, 85L, 22L, 121L, 48L, 105L, 8L, 68L, 123L, 2L, 24L, 38L, 41L, 4L, 59L, 8
7L, 103L, 75L, 83L, 20L, 79L, 41L, 46L, 60L, 45L, 105L, 73L, 46L, 12L, 94L, 14L,
 53L, 55L, 80L, 23L, 96L, 57L, 64L]
```

Decryption :

```
Bob decrypts message sent to him
Decrypted Message           :  [1L, 0L, 1L, 0L, 1L, 1L, 1L]
Decryption took 0.12 seconds
This took 2.07 seconds
>>>
```

Now the results of NTRU KEY EXCHANGE

With parameters, N = 251, p = 3 and q = 128

$$f_i(x) = x^{12} + x^{11} + x^{10} - x^9 + x^8 - x^7 - x^5 + x^4 + x^2 - x - 1$$

$$g_i(x) = x^{10} - x^9 + x^8 - x^7 + x^6 + x^5 - x^3 + x^2 - x - 1$$

$$d_i = 5$$

$$f_j(x) = x^{12} - x^{10} - x^9 - x^8 + x^7 + x^6 + x^5 + x^4 + x^2 - x - 1$$

$$g_j(x) = -x^{10} + x^9 - x^8 + x^7 + x^6 + x^5 - x^3 - x^2 - x + 1$$

$$d_j = 5$$

All the process(NTRU KEY EXCHANGE) took 112.99 sec:

STEP 1: Bob generate H_i and send it to Alice

```
GEN PUBLIC KEY 39.75 seconds
------------------------------------------------------------------STEP1-------------------
------------------------------
Bob Will Generate his Public Key using Parameters
F =  [1, 1, 1, -1, 1, -1, 0, -1, 1, 0, 1, -1, -1]
G =  [1, -1, 1, -1, 1, 1, 0, -1, 1, -1, -1]
d   =  5
Public Key is Generated by Bob:  [121L, 5L, 21L, 74L, 33L, 35L, 59L, 54L, 59L, 122L, 5
0L, 67L, 107L, 45L, 44L, 78L, 83L, 41L, 26L, 13L, 66L, 93L, 41L, 79L, 114L, 73L, 98L,
78L, 115L, 76L, 1L, 79L, 36L, 5L, 99L, 11L, 116L, 21L, 47L, 82L, 74L, 78L, 116L, 60L,
13L, 30L, 8L, 1L, 34L, 108L, 8L, 84L, 36L, 54L, 122L, 116L, 101L, 80L, 46L, 116L, 85L,
94L, 21L, 79L, 100L, 115L, 16L, 17L, 77L, 100L, 74L, 30L, 109L, 105L, 99L, 73L, 76L, 4
2L, 1L, 4L, 5L, 36L, 101L, 30L, 113L, 17L, 62L, 85L, 63L, 114L, 48L, 62L, 126L, 89L, 8
2L, 45L, 17L, 113L, 20L, 19L, 66L, 60L, 61L, 17L, 36L, 47L, 42L, 115L, 103L, 56L, 18L,
0L, 126L, 38L, 117L, 111L, 62L, 96L, 38L, 94L, 66L, 125L, 103L, 77L, 54L, 27L, 112L, 1
18L, 127L, 40L, 34L, 59L, 127L, 43L, 27L, 17L, 69L, 40L, 52L, 4L, 109L, 95L, 0L, 1L, 1
03L, 103L, 73L, 77L, 22L, 95L, 66L, 105L, 55L, 25L, 84L, 109L, 88L, 115L, 14L, 16L, 12
7L, 87L, 44L, 108L, 103L, 28L, 127L, 77L, 106L, 89L, 6L, 112L, 37L, 124L, 33L, 114L, 3
9L, 44L, 70L, 119L, 78L, 16L, 33L, 58L, 99L, 7L, 92L, 43L, 38L, 115L, 69L, 125L, 23L,
48L, 43L, 21L, 49L, 86L, 44L, 61L, 62L, 66L, 124L, 111L, 23L, 14L, 48L, 87L, 62L, 35L,
114L, 99L, 98L, 57L, 116L, 35L, 125L, 51L, 114L, 26L, 19L, 124L, 110L, 103L, 11L, 92L,
80L, 76L, 67L, 109L, 26L, 21L, 57L, 25L, 92L, 86L, 4L, 47L, 52L, 109L, 86L, 123L, 7L,
126L, 46L, 90L, 81L, 32L, 93L, 42L, 100L]
Public key is sending from Alice...
Count =  1
```

STEP 2: Alice takes H_i and create E_J . Also, Alice generates her public key H_J. Sends both H_j and E_j to Bob.

```
Alice Will Generate her Public Key using Parameters
f(x)=  [1, 0, -1, -1, -1, 1, 1, 1, 1, 0, 1, -1, -1]
g(x)=  [-1, 1, -1, 1, 1, 1, 0, -1, -1, -1, 1]
d   =  5
Alice's Random Polynomial  :  [1, 1, -1, -1, -1, -1, -1, 1, -1, 1]
Public Key Hj Generated by Alice:  [87L, 60L, 14L, 59L, 14L, 32L, 102L, 113L, 64L, 95L, 109L,
112L, 12L, 20L, 46L, 60L, 26L, 75L, 30L, 53L, 43L, 98L, 0L, 107L, 71L, 10L, 34L, 5L, 66L, 81L,
 85L, 56L, 125L, 98L, 116L, 82L, 98L, 116L, 33L, 19L, 16L, 12L, 56L, 57L, 53L, 73L, 58L, 16L,
96L, 99L, 95L, 121L, 62L, 83L, 69L, 125L, 127L, 15L, 66L, 73L, 97L, 79L, 54L, 16L, 25L, 53L, 1
03L, 28L, 11L, 18L, 36L, 79L, 106L, 55L, 106L, 7L, 98L, 57L, 55L, 93L, 57L, 42L, 74L, 105L, 53
L, 113L, 0L, 53L, 116L, 39L, 43L, 45L, 4L, 112L, 10L, 83L, 49L, 6L, 3L, 97L, 89L, 17L, 125L, 1
27L, 54L, 113L, 16L, 92L, 5L, 66L, 89L, 22L, 122L, 94L, 124L, 54L, 53L, 10L, 7L, 42L, 42L, 123
L, 95L, 5L, 112L, 30L, 123L, 70L, 56L, 119L, 14L, 64L, 52L, 3L, 6L, 75L, 83L, 46L, 40L, 86L, 6
6L, 74L, 8L, 95L, 70L, 1L, 33L, 44L, 78L, 87L, 41L, 88L, 62L, 47L, 110L, 111L, 91L, 108L, 73L,
 38L, 66L, 28L, 80L, 108L, 119L, 105L, 104L, 86L, 44L, 103L, 123L, 1L, 126L, 16L, 110L, 37L, 1
01L, 61L, 0L, 101L, 25L, 123L, 71L, 33L, 116L, 1L, 17L, 35L, 44L, 54L, 79L, 97L, 85L, 38L, 99L
, 28L, 48L, 89L, 12L, 127L, 46L, 51L, 12L, 63L, 113L, 89L, 31L, 16L, 58L, 127L, 49L, 27L, 70L,
 9L, 4L, 71L, 121L, 43L, 30L, 72L, 48L, 89L, 45L, 32L, 119L, 43L, 39L, 89L, 121L, 15L, 64L, 23
L, 2L, 40L, 39L, 12L, 109L, 108L, 38L, 64L, 106L, 66L, 25L, 82L, 9L, 52L, 40L, 23L, 1L, 15L, 2
0L]
Alice Will Create Ej using parameters:  [1, 0, -1, -1, -1, 1, 1, 1, 1, 0, 1, -1, -1]  and  [1,
 1, -1, -1, -1, -1, -1, 1, -1, 1]
Encrypted Message Ej =  [71L, 100L, 48L, 92L, 7L, 81L, 102L, 25L, 10L, 61L, 99L, 71L, 101L, 83
L, 38L, 25L, 23L, 78L, 59L, 26L, 76L, 72L, 80L, 46L, 98L, 17L, 58L, 77L, 124L, 57L, 86L, 44L,
123L, 5L, 62L, 16L, 94L, 97L, 121L, 17L, 114L, 15L, 81L, 106L, 56L, 93L, 27L, 21L, 66L, 42L, 2
8L, 68L, 114L, 50L, 43L, 95L, 92L, 5L, 27L, 99L, 26L, 112L, 101L, 126L, 120L, 45L, 90L, 69L, 4
8L, 98L, 3L, 103L, 103L, 7L, 46L, 2L, 84L, 27L, 63L, 57L, 0L, 100L, 8L, 51L, 105L, 95L, 29L, 6
8L, 92L, 76L, 1L, 115L, 18L, 3L, 34L, 80L, 13L, 65L, 90L, 55L, 11L, 93L, 70L, 74L, 32L, 120L,
11L, 9L, 84L, 35L, 9L, 101L, 100L, 93L, 44L, 60L, 82L, 5L, 50L, 68L, 96L, 113L, 96L, 127L, 62L
, 12L, 110L, 74L, 75L, 109L, 101L, 15L, 55L, 49L, 44L, 98L, 121L, 127L, 6L, 14L, 33L, 15L, 74L
, 108L, 8L, 106L, 92L, 15L, 67L, 90L, 21L, 93L, 124L, 4L, 3L, 107L, 36L, 94L, 90L, 27L, 64L, 8
6L, 103L, 80L, 49L, 104L, 49L, 5L, 15L, 20L, 55L, 86L, 99L, 47L, 123L, 61L, 87L, 82L, 14L, 66L
, 32L, 114L, 96L, 62L, 8L, 113L, 108L, 109L, 59L, 103L, 94L, 47L, 39L, 107L, 61L, 81L, 30L, 11
3L, 95L, 115L, 90L, 39L, 126L, 89L, 17L, 12L, 105L, 38L, 66L, 58L, 32L, 9L, 115L, 85L, 94L, 81
L, 104L, 120L, 30L, 119L, 10L, 103L, 9L, 127L, 70L, 111L, 72L, 109L, 18L, 97L, 34L, 75L, 96L,
114L, 31L, 33L, 89L, 30L, 73L, 35L, 93L, 105L, 65L, 74L, 28L, 106L, 45L, 18L, 19L, 86L, 126L]
Mesaage and Public Key sendinng from Bob...
Count =  2
```

STEP 3: Bob takes H_j and creates E_i. Sends it to Alice

```
ENTITYI ENCRYPTION TIME 0.04 seconds
Ei:  [93L, 78L, 44L, 26L, 102L, 6L, 88L, 99L, 18L, 64L, 29L, 73L, 81L, 53L, 9L, 121L
, 97L, 55L, 33L, 111L, 101L, 65L, 111L, 74L, 107L, 19L, 15L, 103L, 37L, 13L, 15L, 85
L, 4L, 41L, 14L, 124L, 20L, 121L, 94L, 120L, 87L, 79L, 118L, 103L, 88L, 65L, 47L, 3L
, 116L, 78L, 47L, 22L, 86L, 10L, 44L, 0L, 111L, 58L, 52L, 48L, 100L, 96L, 98L, 121L,
77L, 111L, 61L, 20L, 79L, 118L, 9L, 27L, 35L, 84L, 63L, 7L, 76L, 117L, 121L, 100L, 5
7L, 80L, 16L, 22L, 51L, 119L, 109L, 33L, 48L, 102L, 18L, 87L, 105L, 100L, 33L, 45L,
44L, 61L, 34L, 60L, 116L, 94L, 27L, 50L, 82L, 116L, 17L, 107L, 127L, 22L, 82L, 73L,
92L, 9L, 97L, 108L, 75L, 99L, 21L, 95L, 84L, 85L, 12L, 117L, 23L, 41L, 125L, 117L, 4
4L, 71L, 125L, 20L, 91L, 125L, 113L, 106L, 8L, 86L, 64L, 69L, 97L, 43L, 99L, 59L, 11
5L, 17L, 83L, 39L, 75L, 32L, 71L, 123L, 93L, 95L, 101L, 123L, 37L, 61L, 54L, 19L, 32
L, 14L, 12L, 39L, 102L, 100L, 93L, 7L, 114L, 1L, 2L, 81L, 105L, 13L, 50L, 116L, 79L,
48L, 30L, 114L, 114L, 112L, 71L, 112L, 94L, 64L, 122L, 6L, 96L, 23L, 41L, 105L, 29L,
80L, 17L, 62L, 25L, 13L, 9L, 76L, 65L, 1L, 22L, 87L, 57L, 110L, 69L, 36L, 90L, 122L,
67L, 65L, 51L, 13L, 46L, 104L, 120L, 1L, 89L, 54L, 13L, 113L, 94L, 97L, 102L, 102L,
54L, 122L, 101L, 36L, 12L, 114L, 57L, 21L, 87L, 18L, 52L, 69L, 28L, 47L, 91L, 28L, 9
L, 21L, 51L, 41L, 68L, 67L, 104L, 117L, 101L]
```

STEP 4: Bob uses his own private key and E_j to calculate K_i .

Alice also uses her own private key and E_i to calculate K_j

```
------------------------------------------------------------------STEP4-------------------
-----------------------------
Kj =  [1L, 1L, 0L, 0L, 1L, 2L, 1L, 0L, 0L, 1L, 2L, 1L, 1L, 2L, 2L, 2L, 2L, 2L, 1L, 1L,
2L, 1L, 2L, 2L, 1L]
Count =  4
------------------------------------------------------------------STEP5-------------------
-----------------------------
ENTITYI DECRYPTION TIME 0.06 seconds
Ki =  [1L, 1L, 0L, 0L, 1L, 2L, 1L, 0L, 0L, 1L, 2L, 1L, 1L, 2L, 2L, 2L, 2L, 2L, 1L, 1L,
2L, 1L, 2L, 2L, 1L]
Key Exchange have been completed successfully..
This took 111.28 seconds
```

Since both K_i and K_j values are same, key exchange becomes completed.

We run this codes with different security levels and theirs parameters,

NTRU:

	Key Generation(sec	Encryption(sec)	Decryption(sec)	Time(sec)
Moderate	1.64	0.03	0.12	1.87
Standard	4.76	0.04	0.16	5.06
High	11.32	0.07	0.23	11.81
Highest	34.75	0.08	0.36	35.37

NTRU KEY EXCHANGE

	Time(sec)
Moderate	35.41
Standard	112.99
High	291.50
Highest	839.29

An asymmetric key cryptographic system provides high security in all ways: encryption, decryption and complexity are advanced in NTRU

Method	DES	RSA	NTRU
Approach	Symmetric	Asymmetric	Asymmetric
Encryption	Faster	Slow	Fastest
Decryption	Fastest	Slow	Faster
Key Distribution	Difficult	Easy	Easy
Complexity	O(Log N)	O(N^3)	O(NLogN)
Security	Moderate	Highest	High

NTRU fits into the general framework of a probabilistic cryptosystem. This means that encryption includes a random element, so each message has many possible encryptions. Because encryption and decryption with NTRU are extremely fast, key creation is quick and easy. We note here that NTRU takes $O(N^2)$ operations to encrypt or decrypt a message block of length N, making it considerably faster than the $O(N^3)$ operations required by RSA. Further, NTRU key lengths are $O(N)$, which competes well with the $O(N^2)$ key lengths required by other "fast" public keys systems.

Plain Text Block	$N \log_2 p$ bits
Encrypted Text Block	$N \log_2 q$ bits
Encryption Speed*	$O(N^2)$ operations
Decryption Speed	$O(N^2)$ operations
Message Expansion	$\log_p q$-to-1
Private Key Length	$2N \log_2 p$ bits
Public Key Length	$N \log_2 q$ bits

* Precisely, $4N^2$ additions and N divisions by q with remainder

REFERENCES

[1] Merkle, Ralph C (April 1978). "Secure Communications Over Insecure Channels". Communications of the ACM. 21 (4): 294–299. doi:10.1145/359460.359473

[2] W. Diffie and M. E. Hellman. New directions in cryptography. IEEE Trans. Inform. Theory, 22(6):644–654, 1976.

[3] "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice" (PDF). Retrieved 30 October 2015.

[4] Hoffstein J., Lieman D., Pipher J.,Silverman J. "NTRU: A Public Key Cryptosystem", NTRU Cryptosystems, Inc. (www.ntru.com).

[5] J. Pipher Lectures on the NTRU encryption algorithm and digital signature scheme: Grenoble, June 2002 Brown University, Providence RI 02912

[6] http://people.scs.carleton.ca/~maheshwa/courses/4109/Seminar11/NTRU_presentation.pdf

[7] Xinyu Lei, Xiaofeng Liao "NTRU: A Lattice based Public Key Exchange Protocol" College of Computer Science, Chongqing University, Chongqing 400044, China

[8] Schneier, B Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd ed. New York: Wiley, 1996.

[9] Gauss, C. F. §57 in Disquisitiones Arithmeticae. Leipzig, Germany, 1801. Reprinted New Haven, CT: Yale University Press, 1965.

[10] Nagell, T. "Exponent of an Integer Modulo n" and "The Index Calculus." §31 and 33 in Introduction to Number Theory. New York: Wiley, pp. 102-106 and 111-115, 1951.

[11] Levi A., Özcan M., 'Açık Anahtar Tabanlı Sifreleme Neden Zordur?' Türkiye Bilişim Konferansı 2002

[12] http://www.iam.metu.edu.tr/lectures/IntroCryp. pdf

[13] Yerlikaya T, Buluş E, Buluş N, "Asimetrik Şifreleme Algoritmalarında Anahtar Değişim Sistemleri" bildiri

[14] https://link.springer.com/content/pdf/10.1007%2FBFb0054868.pdf

[15] https://www.iacr.org/workshops/tcc2007/Micciancio.pdf

[16] Phong Q. Nguyen and Jacques Stern "The Two Faces of Lattices in Cryptology"Ecole Normale Sup´ ´ erieure, D´epartement d'Informatique

[17] D. Coppersmith and A. Shamir. Lattice attacks on NTRU. In Proc. of Eurocrypt'97, LNCS. IACR, Springer-Verlag, 1997

[18] 127. V. Shoup. Number Theory C++ Library (NTL) version 3.6. Available at http://www.shoup.net/ntl/.

[19] J. Hoffstein, J. Pipher, and J.H. Silverman. NTRU: a ring based public key cryptosystem. In Proc. of ANTS III, volume 1423 of LNCS, pages 267–288. Springer-Verlag, 1998. Additional information at http://www.ntru.com.

[20] E. Jaulmes and A. Joux. A chosen ciphertext attack on NTRU. In Proc. of Crypto 2000, volume 1880 of LNCS. IACR, Springer-Verlag, 2000.