VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE 4003 - CYBER SECURITY

PROJECT REPORT

PROJECT TITLE: NTRU CRYPTOSYSTEM

GROUP MEMBERS

VIBHOR SHARMA (17BCE2152)

MANISH REDDY (17BCE0098)

SLOT: B2+TB2

FACULTY

PROF. PRAMOD KUMAR MAURYA

INDEXING

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	3
	KEYWORD	3
1.	INTRODUCTION	4
2.	BRIEF BACKGROUND	5
	2.1 Earlier studies on similar problem	5
	2.2. Objectives of the proposed study	6
	2.3 Definition of the problem	6
3.	METHODOLOGY ADOPTED	7
	3.1 Steps of NTRU Encryption and Decryption	7
	3.1.1 Key Generation	7
	3.1.2 NTRU Encryption	8
	3.1.3 NTRU Decryption	8
	3.2 Literature Survey	9
	3.3 Comparison of NTRU with another public key cryptosystem	11
	3.4 Comparison of Key sizes of Keys of NTRU, RSA, ECC	11
	3.5 Tools Used	12
4.	RESULTS	13
	4.1 Screenshot of NTRU Encryption and Decryption Example Code	13
	4.2 Interpretation of Result	14
	4.2.1 How NTRU description works	15
5.	CONCLUSION	16
	5.1 Scope for further study	16
	APPENDIX	17
	REFERENCES	25

ABSTRACT

NTRU is one of the few public key cryptosystems which is supposedly quantum computer resistant. Its security is based on the presumed difficulty of a lattice problem, namely, the shortest vector problem. This project describes the NTRU cryptosystem and its cryptanalysis. The project describes how NTRU is a fast implementing algorithm and can be used as an alternative in the android encryption text. Finally, a comparison of the performance of the NTRU with another public key cryptosystem RSA is presented which describes the fast implementation of the NTRU algorithm as compared to various other cryptosystems. The project consists of a full and deep research and analysis on NTRU algorithm.

KEYWORDS

Here is the list of keywords used in this document:

i. NTRU
ii. RSA
iii. ECC
iv. O(N)
v. NTRUEncrypt

CHAPTER 1

INTRODUCTION

Quantum computers are well known to be able to break cryptosystems that are based on the integer factorization problem or some discrete logarithm problem in polynomial time. This affects RSA and ECC and also number field cryptosystems. The NTRU cryptosystem is an alternative algorithm based on a completely different mathematical problem from RSA and ECC called the closest lattice vector problem. As a result, NTRU is very fast and resistant to quantum computers.

NTRU cryptosystem - nth degree truncated polynomial ring unit, relies on the presumed difficulty of factoring certain polynomials in a truncated polynomial ring into a quotient of two polynomials having very small coefficients. In comparison to widely-known systems such as RSA 2 or ECC 3, the main advantage of NTRU is that its time complexity quadratic order O(N2) in worst case. This system is considered as the fastest cryptosystem, and its strong points are short key size and the speed of encryption and decryption. Since both encryption and decryption use only simple polynomial multiplication, these operations are very fast compared to other asymmetric encryption schemes, such as RSA, ElGamal and elliptic curve cryptography.[1]

The security of NTRU is based on intractability of hard problems in certain type of lattices, called convolutional modular lattices, therefore, it is also categorized as a lattice-based cryptosystem.

CHAPTER 2

BRIEF BACKGROUND

The NTRUEncrypt Public Key Cryptosystem is a relatively new cryptosystem. The first version of the system, which was simply called NTRU, was developed around 1996 by three mathematicians (Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman). In 1996 these mathematicians together with Daniel Lieman founded the NTRU Cryptosystems, Inc. and were given a patent (now expired) on the cryptosystem.

During the last ten years' people have been working on improving the cryptosystem. Since the first presentation of the cryptosystem, some changes were made to improve both the performance of the system and its security. Most performance improvements were focused on speeding up the process. Up till 2005 literature can be found that describes the decryption failures of the NTRUEncrypt.[3]

2.1 Earlier studies on similar problem

As for security, since the first version of the NTRUEncrypt, new parameters have been introduced that seem secure for all currently known attacks and reasonable increase in computation power. Now the system is fully accepted to IEEE P1363 standards under the specifications for lattice-based public-key cryptography (IEEE P1363.1). Because of the speed of the NTRUEncrypt Public Key Cryptosystem and its low memory use, it can be used in applications such as mobile devices and Smart-cards.

In April 2011, NTRUEncrypt was accepted as a X9.98 Standard, for use in the financial services industry. Besides being quantum-computer resistant (at least as of this writing), the encryption/decryption speed is faster than that of other practical cryptosystems, in addition, smaller key sizes make the system attractive. Therefore, any application that requires the 2 aspects of fast performance and/or high-levels of security for the next 10 years would benefit from the NTRU solution.

Currently, NTRU satisfies these crucial aspects of security that apply to payment systems, secure messaging and email, mobile e-commerce, vehicle communications (V2V, V2I), military/aerospace, web browsers and servers, remote backup solutions, online presentations/virtual classrooms, utility meters and cloud providers/data centers.

2.2 Objectives of the proposed study

The objective of the project is to implement NTRU algorithm to encrypt and decrypt a message in python and to perform a comparative analysis on performance of NTRU and another public key cryptosystem.

2.3 Definition of the problem

The problem is to implement the NTRU encryption and decryption algorithm in python.
Also, an analysis is to be performed on performance of NTRU as compared to other public key cryptosystems.

CHAPTER 3

METHODOLOGY ADOPTED

The steps involved in NTRU encryption are:

1. Public Key Generation
2. Encryption
3. Decryption

Further in the project, we will show the comparison of NTRU with other public key cryptosystem and will also show the comparison of key sizes.

3.1 Steps of NTRU Encryption and Decryption

In NTRU cryptosytem, operations are based on objects in a truncated polynomial ring R=Z[X]/(X^N-1), polynomial degree at most N-1:

a0 + a1x + a2x^2 + · · · + aN−1x^(N−1)

Following is the list of keys and parameters used in NTRU:

N - the polynomials in the ring R have degree N-1. (Non-secret)
q - the large modulus to which each coefficient is reduced.(Non secret)
p - the small modulus to which each coefficient is reduced.(Non-secret) f - a polynomial that is the private key. g - a polynomial that is used to generate the public key h from f (Secret but discarded after initial use) h - the public key, also a polynomial r - the random "blinding" polynomial (Secret but discarded after initial use)
d - coefficient[2]

3.1.1 Key Generation:

1st step: User B randomly chooses 2 small polynomials f and g in the R (the ring of truncated polynomials).

Note: 1) The values of these polynomials should kept in a secret.
2) A chosen polynomial must have an inverse.

2nd step: The inverse of f modulo q and the inverse of f modulo p will be computed.

Properties:
f*fq^-1 = 1 (modulo q)
f*fp^-1 = 1 (modulo p)

3rd step: Product of polynomials will be computed: h = p * ((Fq)*g)mod q.

Private key of B: the pair of polynomials f and fp .

Public key of B: the polynomial h .

3.1.2 NTRU Encryption:

User A has a message to transmit:

1st step: Puts the message in the form of polynomial m whose coefficients is chosen modulo p between -p/2 and p/2 (centered lift)

2nd step: Randomly chooses another small polynomial r (to obscure the message).

3rd step: Computes the encrypted message: **e= r*h + m (modulo q)**

3.1.3 NTRU Decryption:

B receives a message e from A and would like to decrypt it.
1st step: Using his private polynomial f he computes a polynomial a= f*e(mod q).
B needs to choose coefficients of a that lie in an interval of length q.

2nd step: B computes the polynomial b=a(mod p).
B reduces each of the coefficients of a modulo p.

3rd step: B uses the other private polynomial fp to compute **c=fp*b(modulo p)**, which is the original message of A.

3.2 Literature Survey:

TABLE 3.1 LITERATURE SURVEY

Author	**Method**	**Purpose**	**Pros**	**Cons**
Hoffstein, J., Pipher, J., & Silverman, J. H. (1998). ***NTRU: A ring-based public key cryptosystem [1]***	Uses ring based NTRU public key cryptosystem	This paper describes a new public key cryptosystem, which we call the NTRU system. The encryption procedure uses a mixing system based on polynomial algebra and reduction modulo two numbers p and q, while the decryption procedure uses an unmixing system whose validity depends on elementary probability theory. The security of the NTRU public key cryptosystem comes from the interaction of the polynomial mixing system with the independence of reduction modulo p and q.	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 compares well with the O(N^2) key lengths required by other "fast" public keys systems such as [8, 4].	It is a newer algorithm and require longer keys and cryptograms.
Coppersmith, D., & Shamir, A. (1997). ***Lattice Attacks on NTRU.[2]***	Lattice method of NTRU	This paper presents a lattice-based attack, which should either find the original secret key f or an alternative key f‘ which can be used in place of f to decrypt ciphertexts with only slightly higher computational complexity	A lattice L is constructed, each of whose elements corresponds to a potential decrypting key f‘; the effectiveness of f’ for decrypting is directly related to the length of the corresponding lattice element.	We have to make f’ very small then only we can decrypt f very easily else it becomes tough
Nayak, R., Sastry, C. V., & Pradhan, J. (2008). ***A matrix formulation for NTRU cryptosystem [3]***	Matrix method of NTRU	This paper proposes a method, which is suitable to send large messages in the form of matrices	This method is more secure since matrices are non commutative. As there is no method to know whether a polynomial is invertible or not we can use this method. Because a matrix is invertible only when it’s determinant is	Using matrix method over the polynomial one may take more time for encryption and decryption because of

			found.	its longer process
Gentry, C. (2001). ***Key Recovery and Message Attacks on NTRU-Composite [4]***	Lattice based attack method of NTRU	This paper presents lattice-based attacks that are especially effective when N is composite. It shows how to use low-dimensional lattices to find a folded version of the private key, where the folded private key has d coefficients with d dividing N. This folded private key can be used either to obtain a folding of the plaintext message, or as partial information to help us recover the entire private key	Using this attack, it is possible to recover entire private keys for the NTRU-256 scheme proposed by Silverman in an average of about 3 minutes. This paper shows that choosing N to be a composite number, especially one with a small factor, significantly reduces the security of the NTRU cryptosystem. Also, this paper shows that that it is possible to recover entire NTRU private keys using lattices of much smaller dimension than was previously thought.	To avoid the presented attacks, N should always has to be chosen to be prime, or to have only large nontrivial factors.
Hoffstein, J., Pipher, J., & Silverman, J. H. (2001). ***NSS: An NTRU Lattice-Based Signature Scheme. [5]***	NSS algorithm method	This paper presents a complete version of NSS and a set of parameters optimized to provide security comparable to RSA 1024 along with high efficiency. This paper then describe the properties of an implementation of this system at these parameters. This paper originally provides a security analysis tailored specifically to these parameters	In order to optimize NSS, the rump session version used disparate sized coefficients whose existence was concealed by allowing p to divide q, which led to a statistical weakness. The use of uniform coefficients and relatively prime values for p and q makes NSS more closely resemble the original NTRU public key cryptosystem, a system that has withstood intense scrutiny since its introduction at CRYPTO 96.	Underlying NTRU is a hard mathematical problem of finding short vectors in certain lattices.

3.3 Comparison of NTRU with another public key cryptosystem:

Public-key cryptography uses two separate keys for encryption and decryption. Even though it is not the same key, the two parts of this key pair are mathematically linked. The public key is used to encrypt plaintext or to verify a digital signature which can be announced to the public. The private key used to decrypt ciphertext or to create a digital signature must be kept secret. Public key algorithm are usually based on the computational complexity of hard problems. For example, the RSA cryptosystem is based on the difficulty of the integer factorization problem, Elliptic curve cryptography (ECC) is based on the difficulty of number theoretic problems involving elliptic curves.

3.4 Comparison of Key sizes of Keys of NTRU, RSA, ECC:

In cryptography, key size or key length is the size measure in bits of the key used in a cryptographic algorithm. The minimum of a key length that is considered strong security is 80 bits. However, the key length of 128 bits is recommended because it offers more security. The private and public keys in RSA and ECC can be chosen from almost equal lengths. The message expansion of NTRU is [n / (n - k)] logp q to 1.

TABLE 3.2 COMPARISON OF NTRU, RSA AND ECC

Security Level(bits)	n	NTRU Key Sizes(bits)	RSA Key Sizes(bits)	ECC Key Sizes(bits)
80	251	2008	1024	163
112	347	3033	2048	224
128	397	3501	3072	256
160	491	4383	4096	320
192	587	5193	7680	384
256	787	7690	15360	521

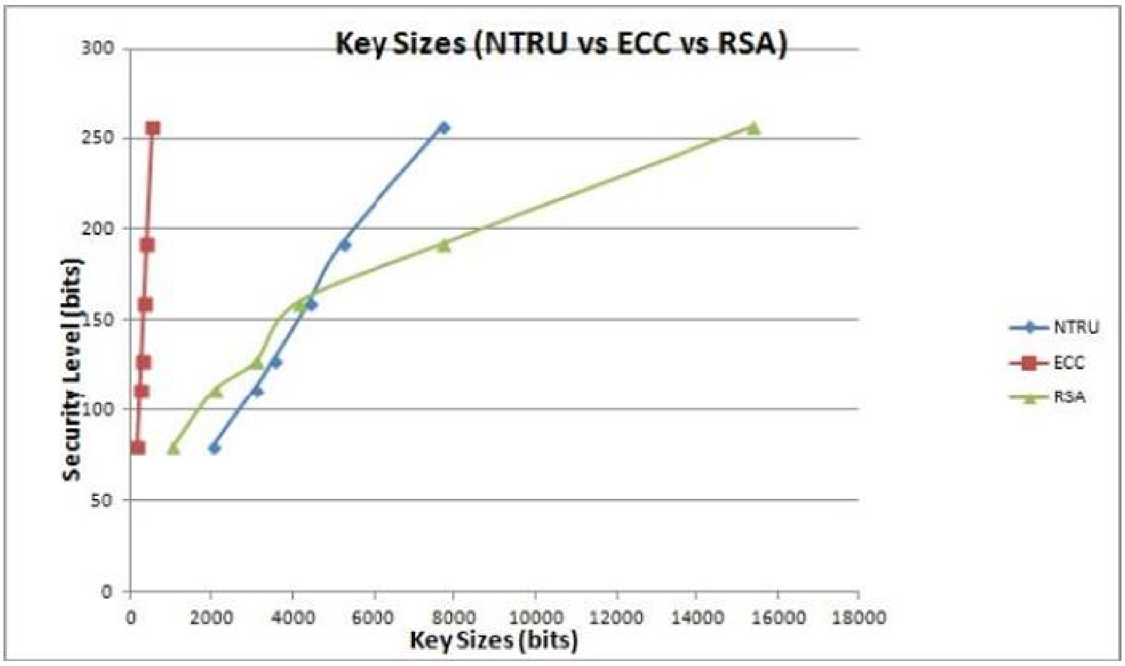


FIGURE 3.1 COMPARISON OF KEYS OF NTRU, RSA AND ECC

3.5 Tools Used:

Spyder was used to implement the algorithm by writing a python code and Microsoft Excel was used to plot the graph of analysis of NTRU encryption and decryption timings for different text sizes.

CHAPTER 4

RESULTS

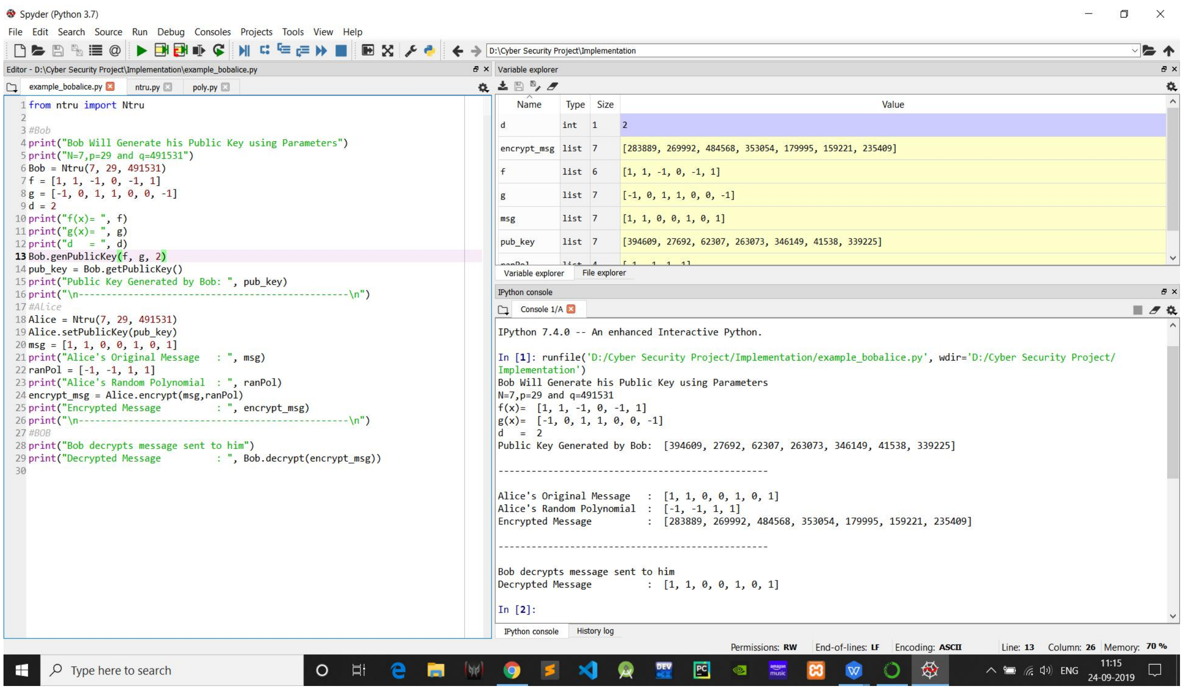
Using the latest suggested parameters, the NTRUEncrypt public key cryptosystem is secure to most attacks. There continues however to be a struggle between performance and security. It is hard to improve the security without slowing down the speed, and vice-versa.

One way to speed up the process without damaging the effectiveness of the algorithm, is to make some changes in the secret key f. First, construct f such that, f= 1 +pF in which F is a small polynomial (i.e. coefficients {-1,0,1}). By constructing f this way, f is invertible mod p. In fact f-1= 1(mod p) , which means that Bob does not have to actually calculate the inverse and that Bob does not have to conduct the second step of decryption. Therefore, constructing f this way saves a lot of time but it does not affect the security of the NTRUEncrypt because it is only easier to find fp but f is still hard to recover. In this case f has coefficients different from -1, 0 or 1, because of the multiplication by p. But because Bob multiplies by p to generate the public key h, and later on reduces the ciphertext modulo p, this will not have an effect on the encryption method. Second, f can be written as the product of multiple polynomials, such that the polynomials have many zero coefficients. This way fewer calculations have to be conducted.

Second, f can be written as the product of multiple polynomials, such that the polynomials have many zero coefficients. This way fewer calculations have to be conducted.

In most commercial applications of the NTRUEncrypt, the parameter N=251 is used. To avoid lattice attacks, brute force attacks and meet-in-the-middle attacks, f and g should have about 72 non-zero coefficients.

4.1 Screenshot of NTRU Encryption and Decryption Example Code:


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Cyber Security Project\Implementation
Editor - D:\Cyber Security Project\Implementation\example_bobalice.py
example_bobalice.py
ntru.py
poly.py
Variable explorer
File explorer
IPython console
Console 1/A
History log
Permissions: RW End-of-lines: LF Encoding: ASCII Line: 13 Column: 26 Memory: 70 %
Type here to search
ENG
11:15
24-09-2019

```python
from ntru import Ntru

#Bob
print("Bob Will Generate his Public Key using Parameters")
print("N=7,p=29 and q=491531")
Bob = Ntru(7, 29, 491531)
f = [1, 1, -1, 0, -1, 1]
g = [-1, 0, 1, 1, 0, 0, -1]
d = 2
print("f(x)= ", f)
print("g(x)= ", g)
print("d   = ", d)
Bob.genPublicKey(f, g, 2)
pub_key = Bob.getPublicKey()
print("Public Key Generated by Bob: ", pub_key)
print("\n-------------------------------------------------\n")
#Alice
Alice = Ntru(7, 29, 491531)
Alice.setPublicKey(pub_key)
msg = [1, 1, 0, 0, 1, 0, 1]
print("Alice's Original Message   : ", msg)
ranPol = [-1, -1, 1, 1]
print("Alice's Random Polynomial  : ", ranPol)
encrypt_msg = Alice.encrypt(msg,ranPol)
print("Encrypted Message          : ", encrypt_msg)
print("\n-------------------------------------------------\n")
#BOB
print("Bob decrypts message sent to him")
print("Decrypted Message          : ", Bob.decrypt(encrypt_msg))
```

Name	Type	Size	Value
d	int	1	2
encrypt_msg	list	7	[283889, 269992, 484568, 353054, 179995, 159221, 235409]
f	list	6	[1, 1, -1, 0, -1, 1]
g	list	7	[-1, 0, 1, 1, 0, 0, -1]
msg	list	7	[1, 1, 0, 0, 1, 0, 1]
pub_key	list	7	[394609, 27692, 62307, 263073, 346149, 41538, 339225]

```
IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/Cyber Security Project/Implementation/example_bobalice.py', wdir='D:/Cyber Security Project/
Implementation')
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, 1, 0, 0, 1, 0, 1]
Alice's Random Polynomial  :  [-1, -1, 1, 1]
Encrypted Message          :  [283889, 269992, 484568, 353054, 179995, 159221, 235409]

-------------------------------------------------

Bob decrypts message sent to him
Decrypted Message          :  [1, 1, 0, 0, 1, 0, 1]

In [2]:
```

FIGURE 4.1 OUPUT OF NTRU IMPLEMENTATION IN PYTHON

4.2 Interpretation of Result:

After successfully implementing the python code of NTRU encryption and decryption, on the basis of the text size of the message to be encrypted, different timings of encryption and decryption were noted for different text sizes and the result is shown beow.

TABLE 4.1 NTRU ENCRYPTION AND DECRYPTION TIMINGS

Text Size	Encryption	Decryption
128 bits	0.0000001	0.0000001
256 bits	0.0000001	0.0549
512 bits	0.05494	0.0549
1K	0.10989	0.0549
2K	0.27472	0.0549
5K	0.65934	0.16484
10K	1.31868	0.361

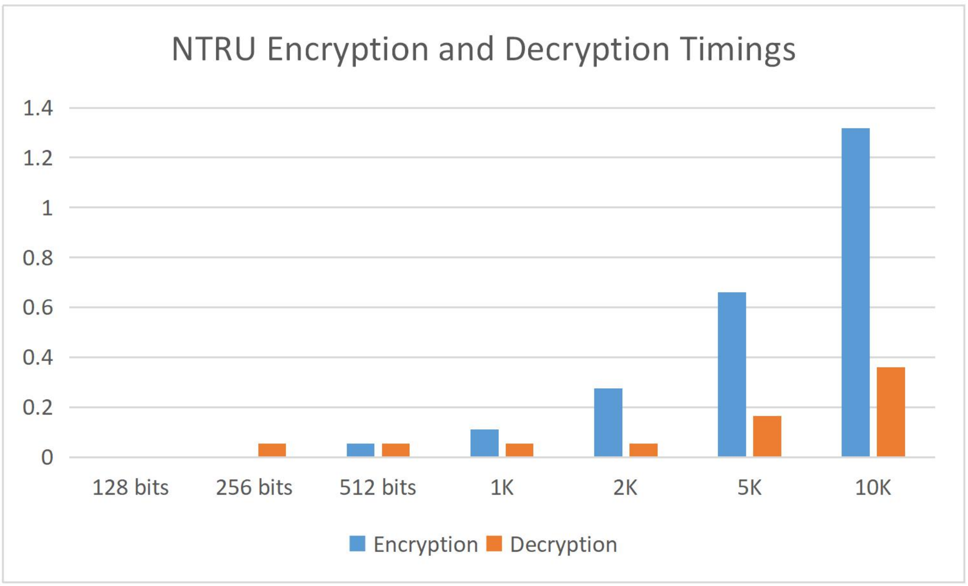


FIGURE 4.2 GRAPH OF NTRU ENCRYPTION AND DECRYPTION TIMINGS

4.2.1 How NTRU description works

If the NTRU parameters (N, p, q, d – public parameters) are chosen to satisfy $q > (6d + 1) p$ this inequality ensures that decryption never fails.

Example: (N, p, q, d) = (7, 3, 41, 2).
According to the example: $41 = q > (6d + 1)p = 39$

According to the latest research the following parameters are considered secure:

TABLE 4.2 PARAMETERS OF NTRU CRYPTOSYSTEM

Parameters	N	q	p
Moderate Security	167	128	3
Standard Security	251	128	3
High Security	347	128	3
Highest Security	503	256	3

CHAPTER 5

CONCLUSION

After successfully implementing the python code for NTRU encryption and decryption, as a programmer it can be understood that decoding the encrypted message will be very difficult for a third user who is trying to break in the system.

After analyzing the performance of NTRU with respect to other public key cryptosystems such as RSA and ECC it was found that it is much more faster and secure.

The first part of this project described the NTRU cryptosystem parameters, private key, public key, encryption and decryption. Next, some of the general attacks against the NTRU cryptosystem were discussed such as alternate private keys, brute force, meet-in-the-middle, multiple transmission and lattice attacks. Following in the project we compared NTRU with RSA.

The performance of RSA was compared with NTRU. In this comparison, it was concluded that NTRU was better than RSA if the security level starts from 192 bits to 256 bits. However, in terms of key generation, encryption, decryption and so on, NTRU out performed RSA.

Thus, through all the analysis we concluded that NTRU is a very potential alternative for android based text encryption and has wide applications in cryptography.

5.1 Scope for further study:

There can be research in the ways of hacking into the cryptosystem and decrypt the encrypted message by a third user. Since it is known that NTRU cryptosystem is resistant to quantum computers, so other ways should be found out to break it so that the faults can be found out and it's security can be improved.

APPENDIX:

This section contains the python code of the implementation of NTRU encryption and decryption.

poly.py

```
# Polynomial Module
from operator import add
from operator import neg    #operator is a module
from operator import mod
from fractions import Fraction as frac    #fractions is a module
from numpy.polynomial import polynomial as P
#numpy is a package

#Helper Functions
# Extended Euclidean Algo for Integers

def egcd(a, b):
        x,y, u,v = 0,1, 1,0
        while a != 0:
                q, r = b//a, b % a
                m, n = x-u * q, y-v * q
                b,a, x,y, u,v = a,r, u,v, m,n
        gcdVal = b
        return gcdVal, x, y



#Modular inverse
#An application of extended GCD algorithm to finding modular inverses:
def modinv(a, m):
        gcdVal, x, y = egcd(a, m)
        if gcdVal != 1:
                return None    # modular inverse does not exist
        else:
                return x % m

#Modulus Function which handles Fractions aswell
def fracMod(f,m):
       [tmp,t1,t2]=egcd(f.denominator,m)
       if tmp!=1:
             print("ERROR GCD of denominator and m is not 1")
             return 0
       else:
             out=modinv(f.denominator,m)*f.numerator % m
        return out

# Resize Adds Leading Zeros to the polynomial vectors
def resize(c1,c2):
    if(len(c1)>len(c2)):
        c2=c2+[0]*(len(c1)-len(c2))
    if(len(c1)<len(c2)):
        c1=c1+[0]*(len(c2)-len(c1))
    return [c1,c2]

# Removes Leading Zeros
def trim(seq):
    if len(seq) == 0:
        return seq
    else:
        for i in range(len(seq) - 1, -1, -1):
            if seq[i] != 0:
                break
        return seq[0:i+1]

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

# Adds two Polynomials
def addPoly(c1,c2):
    [c1,c2]=resize(c1,c2)
    out=list(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)

#Divides two polynomials
def divPoly(N,D):
    N, D = list(map(frac,trim(N))), list(map(frac,trim(D)))
    degN, degD = len(N)-1, len(D)-1
    if(degN>=degD):
        q=[0]*(degN-degD+1)
        while(degN>=degD and N!=[0]):
            d=list(D)
            [d.insert(0,frac(0,1)) for i in range(degN-degD)]
            q[degN-degD]=N[degN]/d[len(d)-1]
            d=list(map(lambda x: x*q[degN-degD],d))
            N=subPoly(N,d)
            degN=len(N)-1
        r=N
    else:
        q=[0]
        r=N
    return [trim(q),trim(r)]

#lambda is for defining anonymous function

# Polynomial Coefficients mod k
def modPoly(c,k):
    if(k==0):
        print("Error in modPoly(c,k). Integer k must be non-zero")
    else:
        return list(map(lambda x: fracMod(x,k),c))

# Centerlift of Polynomial with respect to q
def cenPoly(c,q):
    u=float(q)/float(2)
    l=-u
    c=modPoly(c,q)
    c=list(map(lambda x: mod(x,-q) if x>u else x,c))
    c=list(map(lambda x: mod(x,q) if x<=l else x,c))
    return c

# Extended Euclidean Algorithm
def extEuclidPoly(a,b):
     switch = False
     a=trim(a)
     b=trim(b)
     if len(a)>=len(b):
          a1, b1 = a, b
     else:
          a1, b1 = b, a
```

```
            switch = True
        Q,R=[],[]
        while b1 != [0]:
            [q,r]=divPoly(a1,b1)
            Q.append(q)
            R.append(r)
            a1=b1
            b1=r
        S=[0]*(len(Q)+2)
        T=[0]*(len(Q)+2)
        S[0],S[1],T[0],T[1] = [1],[0],[0],[1]
        for x in range(2, len(S)):
            S[x]=subPoly(S[x-2],multPoly(Q[x-2],S[x-1]))
            T[x]=subPoly(T[x-2],multPoly(Q[x-2],T[x-1]))

        gcdVal=R[-2]
        s_out=S[-2]
        t_out=T[-2]
        ### ADDITIONAL STEPS TO SCALE GCD SUCH THAT LEADING TERM AS COEF
OF 1:
        scaleFactor=gcdVal[len(gcdVal)-1]
        gcdVal=list(map(lambda x:x/scaleFactor,gcdVal))
        s_out=list(map(lambda x:x/scaleFactor,s_out))
        t_out=list(map(lambda x:x/scaleFactor,t_out))
        if switch:
            return [gcdVal,t_out,s_out]
        else:
            return [gcdVal,s_out,t_out]



def isTernary(f,alpha,beta):
    ones=0
    negones=0
    for i in range(0,len(f)):
        if(f[i]==1):
            ones=ones+1
        if(f[i]==-1):
            negones=negones+1
    if (negones+ones)<=len(f) and alpha==ones and beta==negones :
        return True
    else:
        return False
```

ntru.py

```
# Implementation of NTRU encryption
import math
from fractions import gcd
import poly

class Ntru:
    N, p, q, d, f, g, h = None, None, None, None, None, None, None
    f_p, f_q, D = None, None, None
    def __init__(self, N_new, p_new, q_new):
        self.N = N_new
        self.p = p_new
        self.q = q_new
        D = [0] * (self.N + 1)
        D[0] = -1
        D[self.N] = 1
        self.D = D

    def genPublicKey(self, f_new, g_new, d_new):
        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

    def encrypt(self,message,randPol):
        if self.h!=None:

    e_tilda=poly.addPoly(poly.multPoly(poly.multPoly([self.p],randPol),self.h),message)
            e=self.reModulo(e_tilda,self.D,self.q)
            return e
        else:
            print("Cannot Encrypt Message Public Key is not set!")
            print("Cannot Set Public Key manually or Generate it")

    def decryptSQ(self,encryptedMessage):
        F_p_sq=poly.multPoly(self.f_p,self.f_p)
        f_sq=poly.multPoly(self.f,self.f)
        tmp=self.reModulo(poly.multPoly(f_sq,encryptedMessage),self.D,self.q)
        centered=poly.cenPoly(tmp,self.q)
        m1=poly.multPoly(F_p_sq,centered)
        tmp=self.reModulo(m1,self.D,self.p)
        return poly.trim(tmp)

    def decrypt(self,encryptedMessage):
        tmp=self.reModulo(poly.multPoly(self.f,encryptedMessage),self.D,self.q)
        centered=poly.cenPoly(tmp,self.q)
        m1=poly.multPoly(self.f_p,centered)
        tmp=self.reModulo(m1,self.D,self.p)
        return poly.trim(tmp)

    def reModulo(self,num,div,modby):
        [_,remain]=poly.divPoly(num,div)
        return poly.modPoly(remain,modby)

    def printall(self):
        print(self.N)
        print(self.p)
        print(self.q)
        print(self.f)
        print(self.g)
        print(self.h)
        print(self.f_p)
        print(self.f_q)
        print(self.D)

    def isPrime(self):
        if self.N % 2 == 0 and self.N > 2:
            return False
        return all(self.N % i for i in range(3, int(math.sqrt(self.N)) + 1, 2))

    def runTests(self):
        if not self.isPrime() :
            print("Error: N is not prime!")
            return False
        if gcd(self.N,self.p) != 1 :
            print("Error: gcd(N,p) is not 1")
            return False

        if gcd(self.N,self.q)!=1 :
            print("Error: gcd(N,q) is not 1")
            return False

        if self.q<=(6*self.d+1)*self.p:
            print("Error: q is not > (6*d+1)*p")
            return False

        if not poly.isTernary(self.f,self.d+1,self.d):
            print("Error: f does not belong to T(d+1,d)")
            return False

        if not poly.isTernary(self.g,self.d,self.d):
            print("Error: g does not belong to T(d,d)")
            return False

        return(True)
```

example_bobalice.py

```
from ntru import Ntru

#Bob
print("Bob Will Generate his Public Key using Parameters")
print("N=7,p=29 and q=491531")
Bob = Ntru(7, 29, 491531)
f = [1, 1, -1, 0, -1, 1]
g = [-1, 0, 1, 1, 0, 0, -1]
d = 2
print("f(x)= ", f)
print("g(x)= ", g)
print("d    = ", d)
Bob.genPublicKey(f, g, 2)
pub_key = Bob.getPublicKey()
print("Public Key Generated by Bob: ", pub_key)
print("\n----------------------------------------------\n")
#Alice
Alice = Ntru(7, 29, 491531)
```

```
Alice.setPublicKey(pub_key)
msg = [1, 1, 0, 0, 1, 0, 1]
print("Alice's Original Message      : ", msg)
ranPol = [-1, -1, 1, 1]
print("Alice's Random Polynomial    : ", ranPol)
encrypt_msg = Alice.encrypt(msg,ranPol)
print("Encrypted Message                  : ", encrypt_msg)
print("\n-----------------------------------------------\n")
#BOB
print("Bob decrypts message sent to him")
print("Decrypted Message                  : ", Bob.decrypt(encrypt_msg))
```

REFERENCES

[1] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, “NTRU: A Ring-Based Public Key Cryptosystem”, June 1998.

[2] Don Coppersmith, Adi Shamir, “Lattice Attacks on NTRU”, International Conference on the Theory and Applications of Cryptographic Techniques, 1997: Advances in Cryptology-EUROCRYPT ’97 pp 52-61

[3] Nayak, R., Sastry, C. V., & Pradhan, J, “A matrix formulation for NTRU cryptosystem”, 2008.

[4] Craig Gentry, “Key Recovery and Message Attacks on NTRU-Composite”, 2001.

[5] Hoffstein, J., Pipher, J., & Silverman, J. H., “NSS: An NTRU Lattice-Based Signature Scheme”, 2001.

Web References:

[1] https://en.wikipedia.org/wiki/NTRU

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

[3] https://nitaj.users.lmno.cnrs.fr/ntru3final.pdf