Vellore Institute of Technology

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

CSE3502-INFORMATION SECURITY MANAGEMENT

J COMPONENT REPORT

Slot – F1

Title – Chat Encryption using NTRU Algorithm

Team Members: -

Puneet Purohit 20BCE0191

Jayanth T 20BCE0967

Vinay Madana 20BCE2385

Vignesh S R 20BKT0047

Submitted to: -

Dr. Anand M

Associate Professor Sr.

SCOPE

Vellore Institute of Technology, Vellore

Abstract:

These days, security is essential for all network applications. Security is provided in numerous sectors by a variety of methods using a variety of techniques and algorithms. Although NTRU is a quick encryption method that is used in many sectors, it has not yet been included to chat applications. The NTRU method is regarded as the fastest and finest technique for securing data transported via networks. Lattice-based cryptography is used in the patented and open-source public-key cryptosystem known as NTRU to encrypt and decode information. Several applications employ large databases (chats) to store the data and are concurrently available by many users from everywhere via mobile networks. Implementation of the NTRU algorithm for security of such a chat-related application shown that this algorithm offers the best security. Files are encrypted and decrypted using the server's generated keys, which are then discarded after the use, to prevent any data storage attacks.

Introduction:

Chat messages are often brief so that other users may answer quickly and conserve time and other resources. Online chat can include voice-video chat, multicast communications from one sender to many recipients, point-to-point communications, and more. It can also be an element of a web conferencing service. Two of the top cryptographers in the world, Don Coppersmith and Adi Shamir, created the NTRU technology. The techniques of lattice reduction were suggested as the most effective method of attacking the NTRU cryptosystem.

The NTRU algorithm is completely based out on the mathematical polynomial series in order to encrypt and decrypt the data. It has a lattice-based cryptography approach- A lattice is a collection of points in n-dimensional space that creates a regular grid and is used in lattice-based cryptography. A collection of basis vectors is used to construct the lattice, and each point's definition for the lattice is an integer vector. The direction and distance between the lattice points are determined by the basic's vectors. The difficulty of locating a short vector in the lattice from which the public and private keys are formed is the foundation of NTRU's security. It is well known that finding a short vector in a high-dimensional lattice is a challenging computer task, and that it is immune to assaults utilising the most recent cutting-edge methods.

Also, the NTRU algorithm provides better integration and interoperability across the applications also faster key generation than usually used algorithms. The quantum attack which could be possibly break the NTRU is Shor's algorithm. Large integer factors can be found using a quantum technique, as can several other mathematical conundrums like the discrete logarithm issue. The fundamental idea of Shor's technique is to accelerate factoring by doing a large number of concurrent operations on a quantum computer.

Literature Survey:

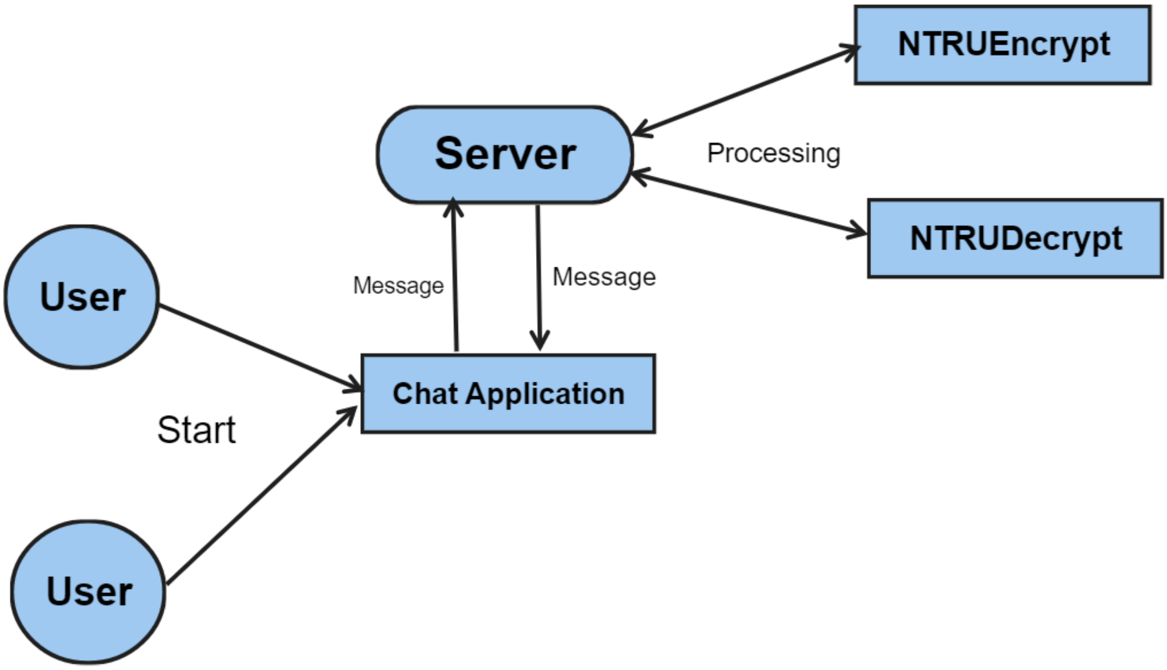
Papers	Takeaways
SHOR’S FACTORING ALGORITHM AND MODERN CRYPTOGRAPHY - AN ILLUSTRATION OF THE CAPABILITIES INHERENT IN QUANTUM COMPUTERS , EDWARD GERJUOY, JUNE 2005 , AMERICAN ASSOCIATION OF PHYSICS TEACHERS	The RSA encryption protocol is explained in this paper, as well as the various quantum computer manipulations that make up the Shor algorithm, how the Shor algorithm performs factoring, and the precise sense in which a quantum computer using Shor's algorithm can be said to factor very large numbers with less computational effort than a classical computer. It is clear that factoring N necessitates a large number of iterations of the procedure. The results show that the likelihood of attaining a successful factorization on a single run is roughly twice as high as previously thought
QUANTUM RESISTANT PUBLIC KEY CRYPTOGRAPHY: A SURVEY, RAY A. PERLNER, DAVID A. COOPER, APRIL 2009, NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY.	They provided a survey of some of the public key cryptographic algorithms that have been created that, while not widely used, are believed to be resistant to quantum computing-based attacks, and discussed some of the issues that protocol designers may need to take into account if these algorithms are needed in the future.
QUANTUM ALGORITHMS FOR COMPUTING SHORT DISCRETE LOGARITHMS AND FACTORING RSA INTEGERS, MARTIN EKER AND JOHAN H, JUNE 2017, SPRINGER INTERNATIONAL	They explain how key problems like factoring RSA integers and determining group order under side information may be recast as small discrete logarithm problems in this study. This results in a less difficult procedure for factoring RSA integers than Shor's general factoring algorithm, in the sense that it places fewer demands on the quantum computer. The primary challenge in both their and Shor's algorithms is computing a modular exponentiation in superposition. When factoring an n^2bit integer, Shor's approach uses an exponent of length 2n bits, but their algorithm uses slightly more than n/2 bits.
SPEED RECORDS FOR NTRU, JENS HERMANS, FREDERIK VERCAUTEREN, AND BART PRENEEL, MARCH 2010, SPRINGER	For the first time, NTRU Encrypt is implemented on a GPU utilizing the CUDA platform in this paper. This operation lends itself excellently to parallelization, as illustrated, and performs exceedingly well when compared to similar security levels for ECC and RSA, with speedups of three to five orders of magnitude. The goal is to achieve a high throughput, which is achieved by conducting a large number of encryptions and decryptions in parallel. At a security level of 256 bits, a current GTX280 GPU can achieve a throughput of up to 200 000 encryptions per second. This results in a theoretical data

	throughput of 47.8 megabytes per second. When compared to a symmetric cipher (a comparison that isn't often made), this is only about 20 times slower than a contemporary AES implementation on a GPU
THE POST-QUANTUM SIGNAL PROTOCOL SECURE CHAT IN A QUANTUM WORLD INES DUITS, FEBRUARY, 2019, UNIVERSITY OF TWENTE	For messaging apps like WhatsApp, Skype, Facebook private Messenger, Google Allo, and Signal, the Signal Protocol enables end-to-end encryption, forward secrecy, backward secrecy, authentication, and deniability. The ECDH Curve25519 key exchanges and SHA-512 key derivation are used in the Signal Protocol to accomplish this. The ECDH key exchange, on the other hand, is not quantum-safe; if opponents had a quantum computer, they could gain the key and read along. They put ten various post quantum key exchange mechanisms (KEMs) to the test, as well as the Diffie-Hellman algorithm based on post quantum super singular isogeny (SIDH). They also looked at various variants of a partially post-quantum Signal Protocol.

Proposed Architecture:

An NTRU-based chat application's suggested architecture is provided below:

1. User Registration: When the application server starts it asks for a nickname. Once the user enters a name it opens the chat window. Multiple users can join the same room and send messages to each other.
2. Encryption: The recipient's public key is used to encrypt the message before it is sent by the user. The message can then be unlocked with the recipient's private key.
3. Decryption: A user's private key is used to decode a communication once they receive it. The legitimacy of the message is confirmed using the sender's public key.
4. Message Storage: Before they are transmitted to the recipient, encrypted communications are kept on the server. The message is removed from the server after it has been delivered.
5. User Interface: For the UI we have used the Tkinter library of python for creating chat application and messaging.



Innovation:

The recent chat applications aren’t resistance to quantum computing attacks, we are incorporating the NTRU for the chats. It provides relatively less computational overhead as compared to other public key cryptosystems and makes it suitable for resource constrained environment. The time for key generation and exchange is also less due to some mathematical optimizations and memory management.

Sample Code:

NTRU.py

```
from . import poly

class NTRU:

    def __init__(self, N, p, q, f, g):

        self.N= N

        self.p = p

        self.q = q

        self.f = f

        self.g = g

        print("==== Generating public and private keys ====")

        print("Values used:")

        print(" N=", N)

        print(" p=", p)

        print(" q=", q)

        print("========")

        print("\nSender pick two polynomials (g and f):")

        print(" f(x)=", f)

        print(" g(x)=", g)

    def gen_keys(self):

        D = [0] * (self.N + 1)

        D[0] = -1

        D[self.N] = 1

        print("\n====Determining F_p and F_q ===")

        [gcd_f, s_f, t_f] = poly.extEuclidPoly(self.f, D)
        self.f_p = poly.modPoly(s_f, self.p)
        self.f_q = poly.modPoly(s_f, self.q)
        print("F_p:", self.f_p)
        print("F_q:", self.f_q)
        x = poly.multPoly(self.f_q, self.g)
        self.h = poly.reModulo(x, D, self.q)
        # return h, self.f, f_p
    def get_pubkey(self):
        return self.h
    def get_privkeys(self):
        return self.f, self.f_p
def encrypt(message, p, q, D, h):
    print("Original message:", message)
    msg1 = list(message)
    for i in range(len(msg1)):
        msg1[i] = ord(msg1[i])
    print(msg1)
    randPol = [-1, -1, 1, 1]
    encmsg = []
    l = []
    for i in msg1:
        print('----', i, '----')
        msg = poly.DecimalToBinary(i)
        print("Sender's Message:\t", msg)
        l.append(len(msg))
        print("Random:\t\t\t", randPol)
        e_tilda = poly.addPoly(poly.multPoly(poly.multPoly([p], randPol),h), msg)
        e = poly.reModulo(e_tilda, D, q)
        encmsg.append(e)
        print("Encrypted message:\t", e)
    print('\n\nenc', encmsg)
    return encmsg, l
def decrypt(encmsg, f, f_p, p, q, D, l):
    decmsg = []
    j = 0
    for e in encmsg:
        tmp = poly.reModulo(poly.multPoly(f, e), D, q)
        centered = poly.cenPoly(tmp, q)
        m1 = poly.multPoly(f_p, centered)
        tmp = poly.reModulo(m1, D, p)
        print("Decrypted message:\t", tmp[:l[j]])
        decmsg.append(tmp[:l[j]])
        j = j + 1
    print('dec', decmsg)
    fm = []
    print("\n\n--start--")
    for i in decmsg:
        decmsgpart = poly.binaryToDecimal(int(".join(list(map(str, i)))))
        print(i, decmsgpart)
        fm.append(decmsgpart)
    print("\n\n")
    print('--decrypt msg\n\n', ".join(list(map(chr, fm))))
    message = ".join(list(map(chr, fm)))
    return message
```

poly.py

```
from operator import add
from operator import neg
from operator import mod
from fractions import Fraction as frac
from . import fracModulo
```

```
def binaryToDecimal(binary):
    binary1 = binary
    decimal, i, n = 0, 0, 0
    while binary != 0:
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary // 10
        i += 1
    return decimal
def DecimalToBinary(num):
    return list(map(int, list(bin(num).replace("0b", ""))))
def modPoly(c, k):
    if (k == 0):
        print("Error in modPoly(c,k). Integer k must be non-zero")
    else:
        return [fracModulo.fracMod(x, k) for x in c]
def subPoly(c1, c2):
    [c1, c2] = resize(c1, c2)
    c2 = list(map(neg, c2))
    out = list(map(add, c1, c2))
    return trim(out)
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)
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]
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]
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[len(R) - 2]
    s_out = S[len(S) - 2]
    t_out = T[len(T) - 2]
    scaleFactor = gcdVal[len(gcdVal) - 1]
    gcdVal = [x / scaleFactor for x in gcdVal]
    s_out = [x / scaleFactor for x in s_out]
    t_out = [x / scaleFactor for x in t_out]
    if switch:
        return [gcdVal, t_out, s_out]
    else:
        return [gcdVal, s_out, t_out]
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 = [x * q[degN - degD] for x in d]
            N = subPoly(N, d)
            degN = len(N) - 1
        r = N
    else:
        q = [0]
        r = N
    return [trim(q), trim(r)]
def addPoly(c1, c2):
    [c1, c2] = resize(c1, c2)
    out = list(map(add, c1, c2))
    return trim(out)
def cenPoly(c, q):
    u = float(q) / float(2)
    l = -u
    c = modPoly(c, q)
    c = [mod(x, -q) if x > u else x for x in c]
    c = [mod(x, q) if x <= l else x for x in c]
    return c
def reModulo(num, div, modby):
    [_, remain] = divPoly(num, div)
    return modPoly(remain, modby)
```

fracModulo.py

```
from fractions import Fraction as frac
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
def modinv(a, m):
    gcdVal, x, y = egcd(a, m)
    if gcdVal != 1:
        return None
    else:
        return x % m
def fracMod(f, m):
    [tmp, t1, t2] = egcd(f.denominator, m)
    if tmp != 1:
        print("ERROR GCD of denominator and m is not 1")
        1 / 0
    else:
        out = modinv(f.denominator, m) * f.numerator % m
        return out
```

Implementation:

Polynomials are truncated under the ring R = Z[X]/(X^N-1)

Three integer parameters N, p and q are assumed with N-1 being the maximum degree of polynomial in the NTRUEncrypt scheme.

Two polynomial pairs f and g are required each of whose coefficients are in the set {-1,0,1}, such that number of 1s as coefficients of g should be equal to number of –1s but they should not be equal in case of f. The polynomial f must satisfy the additional requirement that the inverses modulo q (fq) and modulo p (fp)exist, which means that: - f.f = 1 (mod q)

$$\boxed{\begin{array}{l} f.f_p = 1 \ \ (\mathrm{mod}\ p) \\ \\ f.f_q = 1 \ \ (\mathrm{mod}\ q) \end{array}}$$

The polynomials f, g and fp are B’s private key. The public key h of B is calculated by: -

$$h = \ pf_q.g \ \ (\mathrm{mod}\ q)$$

Encryption

The sender will send the message in the form of a polynomial m whose coefficients will lie in the range [-p/2,p/2]. Message polynomial can be represented in binary format. The sender is required to choose a random polynomial r (termed as blinding value) such that it has coefficients, though these coefficients are not restricted to the set {-1,0,1}. This polynomial must be kept as a secret. Using public key of B, r and m, the encrypted message e is obtained as:

$$e = r.h + m \ \ (\mathrm{mod}\ q)$$

Decryption

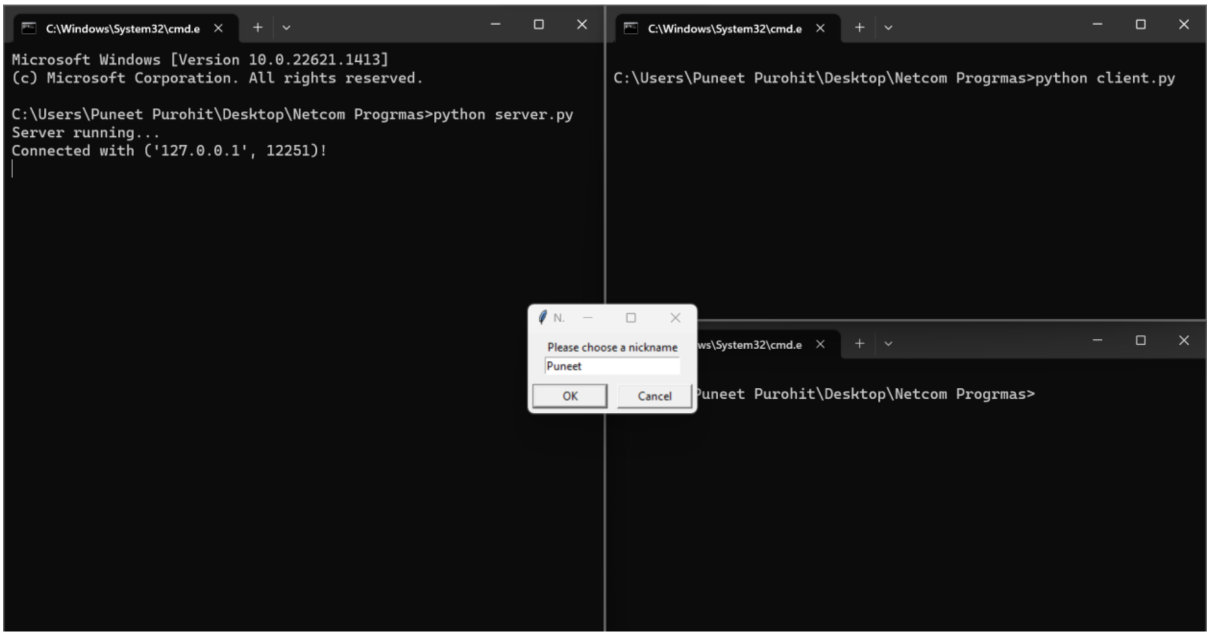
Receiver decrypts the encrypted message e using his/her private key f and performing the following operations to get the decrypted message c which is equivalent to m (mod p): -

$$a = f.e \ \ (\mathrm{mod}\ q)$$

$$b = a \ \ (\mathrm{mod}\ p)$$

$$c = f_p.b$$

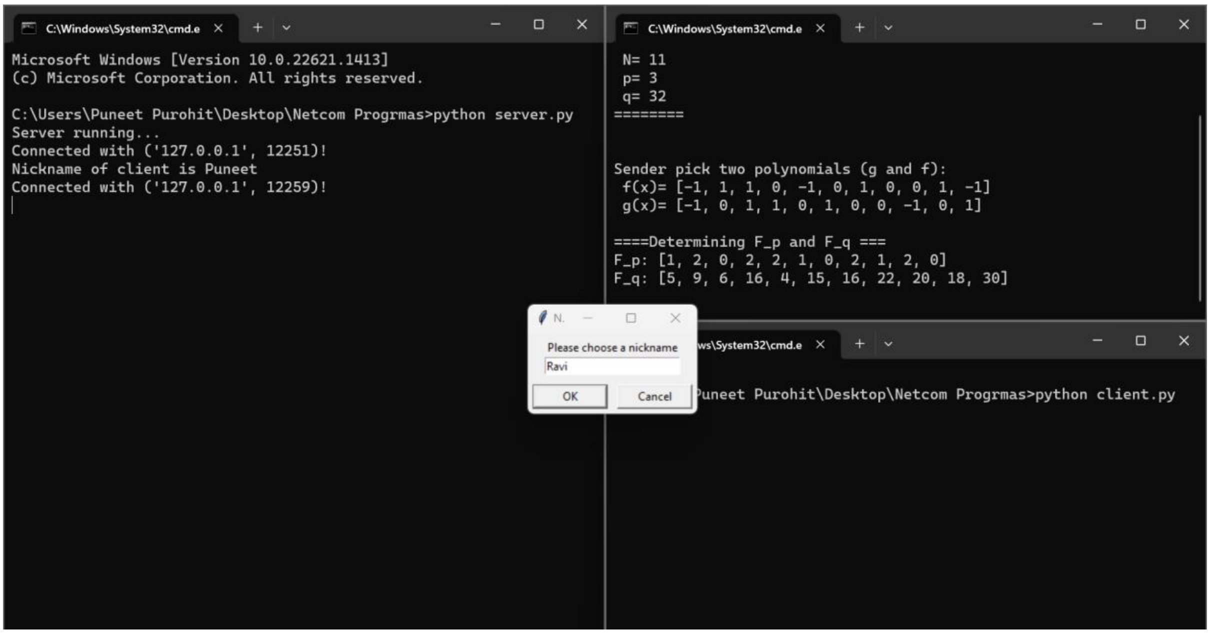
Snapshot Of Implementations:



```
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Puneet Purohit\Desktop\Netcom Progrmas>python server.py
Server running...
Connected with ('127.0.0.1', 12251)!
```

```
C:\Users\Puneet Purohit\Desktop\Netcom Progrmas>python client.py
```

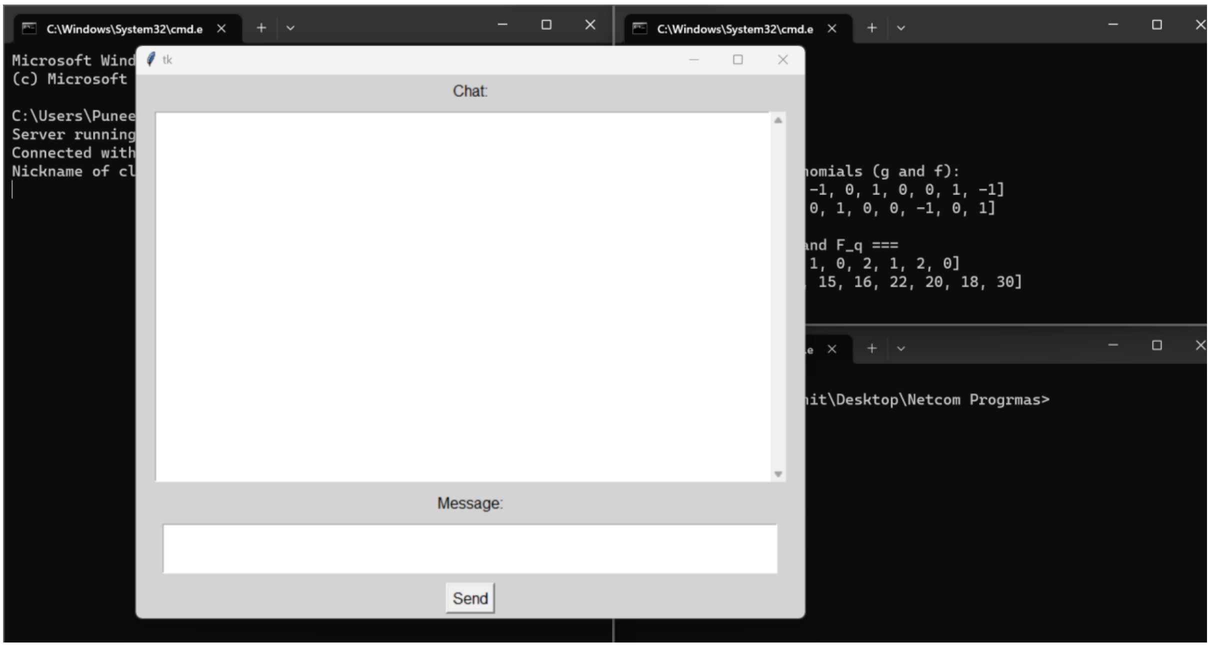
```
Puneet Purohit\Desktop\Netcom Progrmas>
```



```
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Puneet Purohit\Desktop\Netcom Progrmas>python server.py
Server running...
Connected with ('127.0.0.1', 12251)!
Nickname of client is Puneet
Connected with ('127.0.0.1', 12259)!
```

```
 N= 11
 p= 3
 q= 32
========


Sender pick two polynomials (g and f):
 f(x)= [-1, 1, 1, 0, -1, 0, 1, 0, 0, 1, -1]
 g(x)= [-1, 0, 1, 1, 0, 1, 0, 0, -1, 0, 1]

====Determining F_p and F_q ===
F_p: [1, 2, 0, 2, 2, 1, 0, 2, 1, 2, 0]
F_q: [5, 9, 6, 16, 4, 15, 16, 22, 20, 18, 30]
```

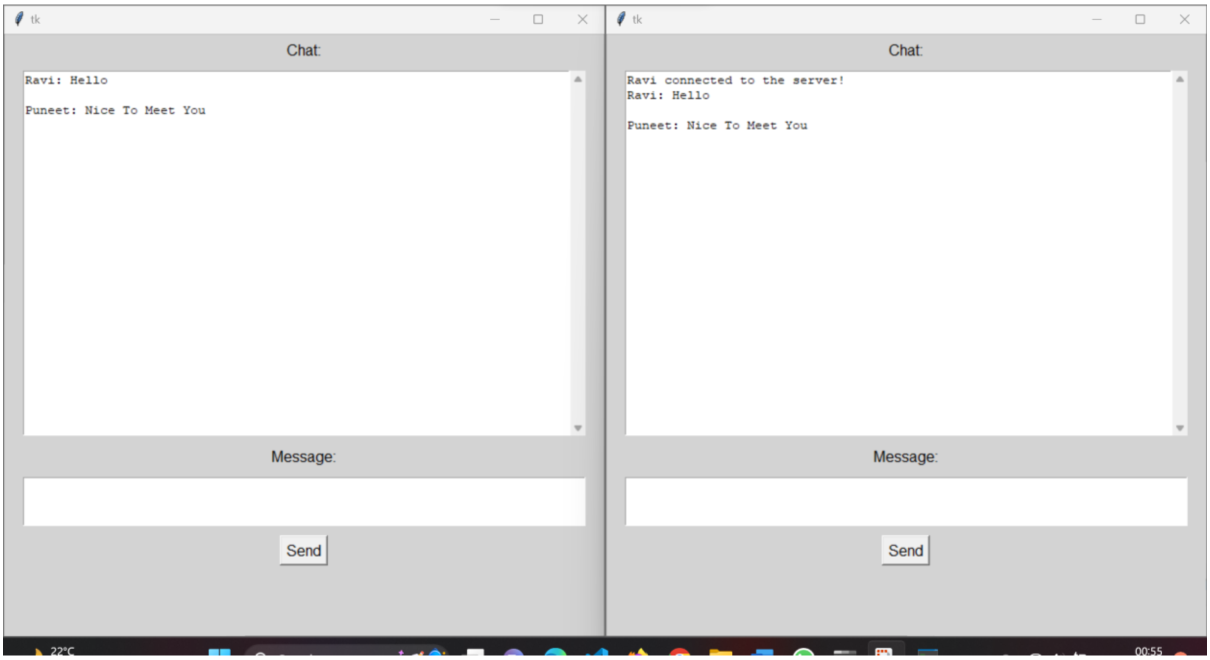
```
Puneet Purohit\Desktop\Netcom Progrmas>python client.py
```



```
Microsoft Wind
(c) Microsoft

C:\Users\Punee
Server running
Connected with
Nickname of cl
```

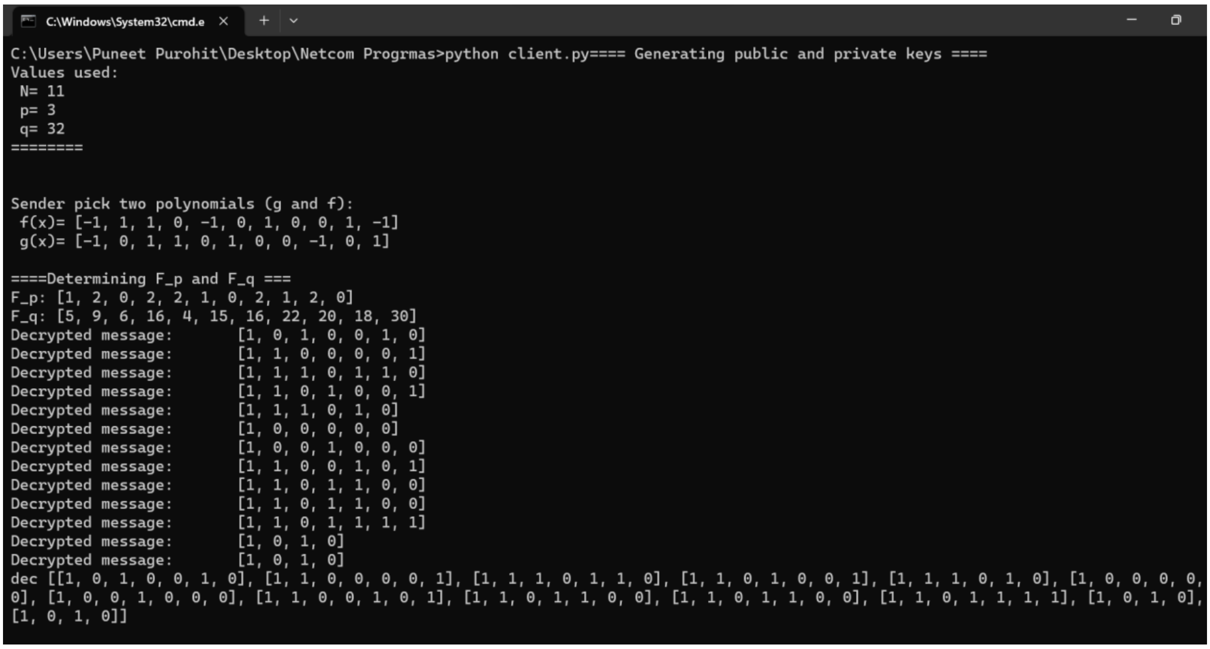
```
nomials (g and f):
 -1, 0, 1, 0, 0, 1, -1]
 0, 1, 0, 0, -1, 0, 1]

and F_q ===
 1, 0, 2, 1, 2, 0]
 15, 16, 22, 20, 18, 30]
```

```
nit\Desktop\Netcom Progrmas>
```

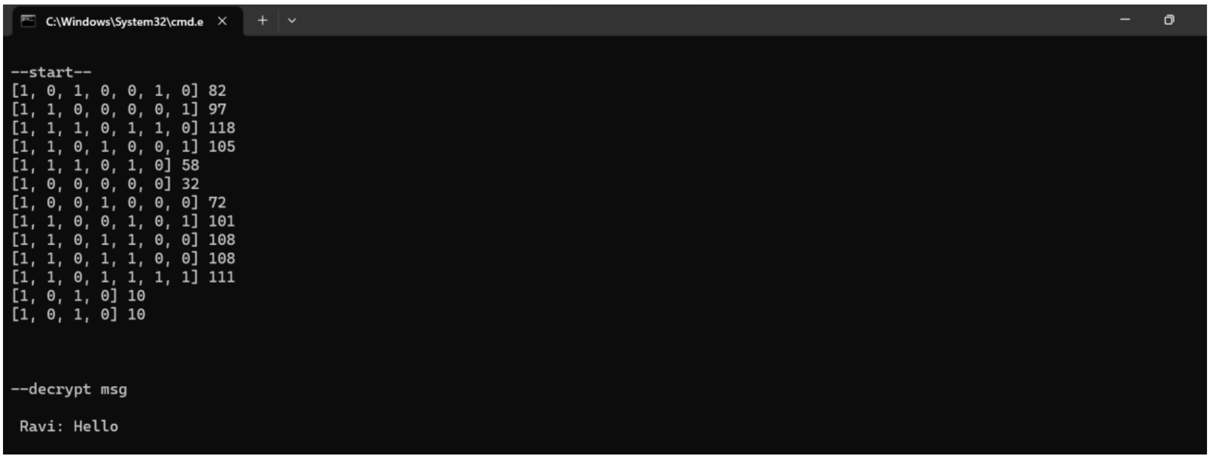


```
Ravi: Hello

Puneet: Nice To Meet You
```

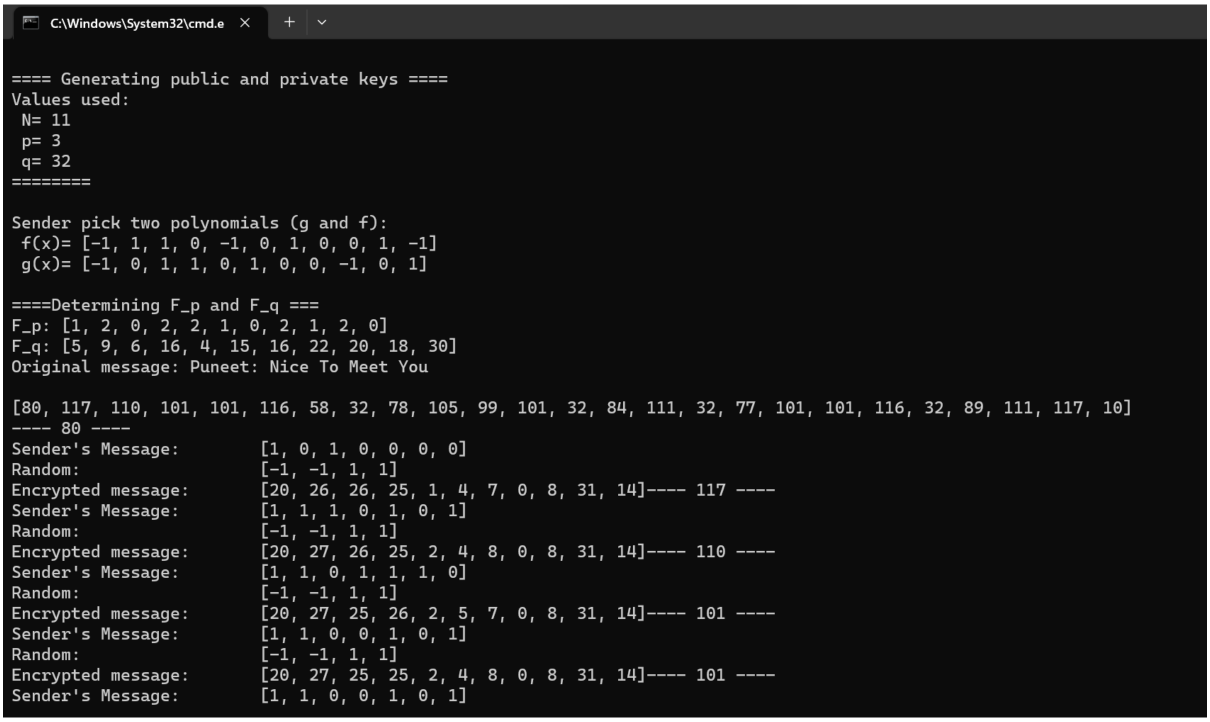
```
Ravi connected to the server!
Ravi: Hello

Puneet: Nice To Meet You
```



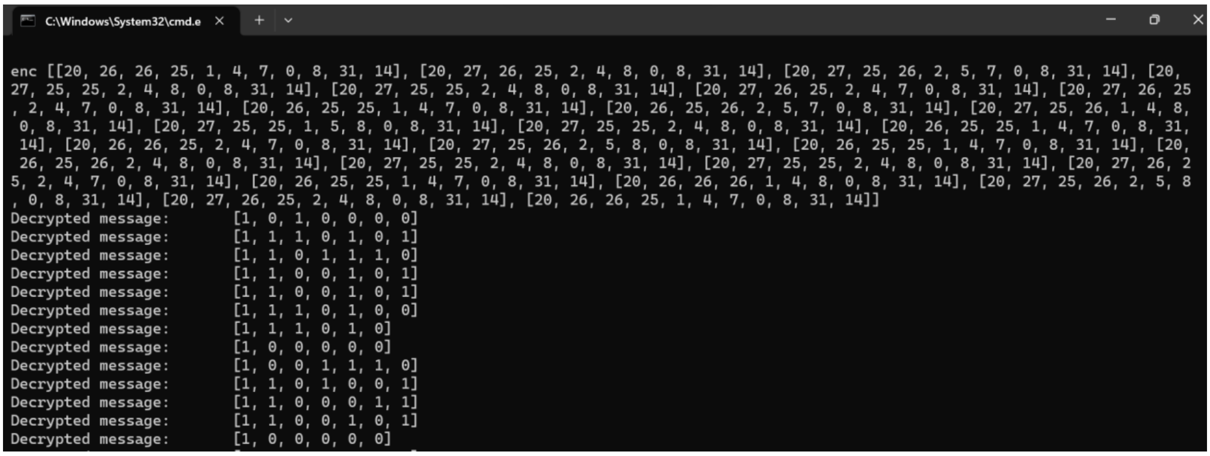
```
C:\Users\Puneet Purohit\Desktop\Netcom Progrmas>python client.py==== Generating public and private keys ====
Values used:
 N= 11
 p= 3
 q= 32
========


Sender pick two polynomials (g and f):
 f(x)= [-1, 1, 1, 0, -1, 0, 1, 0, 0, 1, -1]
 g(x)= [-1, 0, 1, 1, 0, 1, 0, 0, -1, 0, 1]

====Determining F_p and F_q ===
F_p: [1, 2, 0, 2, 2, 1, 0, 2, 1, 2, 0]
F_q: [5, 9, 6, 16, 4, 15, 16, 22, 20, 18, 30]
Decrypted message:       [1, 0, 1, 0, 0, 1, 0]
Decrypted message:       [1, 1, 0, 0, 0, 0, 1]
Decrypted message:       [1, 1, 1, 0, 1, 1, 0]
Decrypted message:       [1, 1, 0, 1, 0, 0, 1]
Decrypted message:       [1, 1, 1, 0, 1, 0]
Decrypted message:       [1, 0, 0, 0, 0, 0]
Decrypted message:       [1, 0, 0, 1, 0, 0, 0]
Decrypted message:       [1, 1, 0, 0, 1, 0, 1]
Decrypted message:       [1, 1, 0, 1, 1, 0, 0]
Decrypted message:       [1, 1, 0, 1, 1, 0, 0]
Decrypted message:       [1, 1, 0, 1, 1, 1, 1]
Decrypted message:       [1, 0, 1, 0]
Decrypted message:       [1, 0, 1, 0]
dec [[1, 0, 1, 0, 0, 1, 0], [1, 1, 0, 0, 0, 0, 1], [1, 1, 1, 0, 1, 1, 0], [1, 1, 0, 1, 0, 0, 1], [1, 1, 1, 0, 1, 0], [1, 0, 0, 0, 0,
0], [1, 0, 0, 1, 0, 0, 0], [1, 1, 0, 0, 1, 0, 1], [1, 1, 0, 1, 1, 0, 0], [1, 1, 0, 1, 1, 0, 0], [1, 1, 0, 1, 1, 1, 1], [1, 0, 1, 0],
[1, 0, 1, 0]]
```



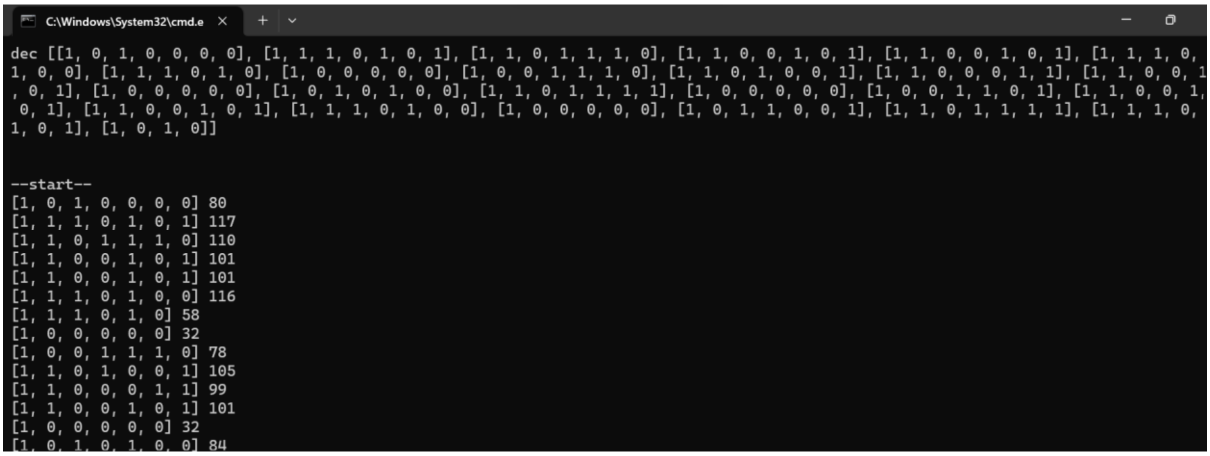
```
--start--
[1, 0, 1, 0, 0, 1, 0] 82
[1, 1, 0, 0, 0, 0, 1] 97
[1, 1, 1, 0, 1, 1, 0] 118
[1, 1, 0, 1, 0, 0, 1] 105
[1, 1, 1, 0, 1, 0] 58
[1, 0, 0, 0, 0, 0] 32
[1, 0, 0, 1, 0, 0, 0] 72
[1, 1, 0, 0, 1, 0, 1] 101
[1, 1, 0, 1, 1, 0, 0] 108
[1, 1, 0, 1, 1, 0, 0] 108
[1, 1, 0, 1, 1, 1, 1] 111
[1, 0, 1, 0] 10
[1, 0, 1, 0] 10



--decrypt msg

 Ravi: Hello
```



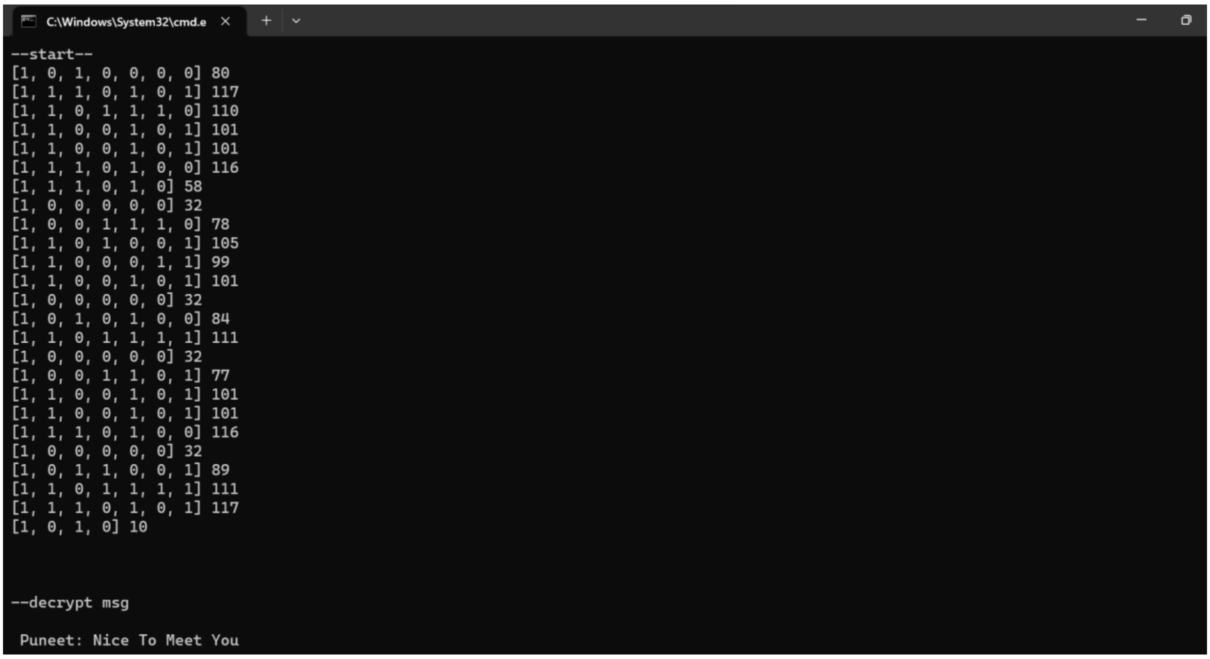
```
==== Generating public and private keys ====
Values used:
 N= 11
 p= 3
 q= 32
========

Sender pick two polynomials (g and f):
 f(x)= [-1, 1, 1, 0, -1, 0, 1, 0, 0, 1, -1]
 g(x)= [-1, 0, 1, 1, 0, 1, 0, 0, -1, 0, 1]

====Determining F_p and F_q ===
F_p: [1, 2, 0, 2, 2, 1, 0, 2, 1, 2, 0]
F_q: [5, 9, 6, 16, 4, 15, 16, 22, 20, 18, 30]
Original message: Puneet: Nice To Meet You

[80, 117, 110, 101, 101, 116, 58, 32, 78, 105, 99, 101, 32, 84, 111, 32, 77, 101, 101, 116, 32, 89, 111, 117, 10]
---- 80 ----
Sender's Message:        [1, 0, 1, 0, 0, 0, 0]
Random:                  [-1, -1, 1, 1]
Encrypted message:       [20, 26, 26, 25, 1, 4, 7, 0, 8, 31, 14]---- 117 ----
Sender's Message:        [1, 1, 1, 0, 1, 0, 1]
Random:                  [-1, -1, 1, 1]
Encrypted message:       [20, 27, 26, 25, 2, 4, 8, 0, 8, 31, 14]---- 110 ----
Sender's Message:        [1, 1, 0, 1, 1, 1, 0]
Random:                  [-1, -1, 1, 1]
Encrypted message:       [20, 27, 25, 26, 2, 5, 7, 0, 8, 31, 14]---- 101 ----
Sender's Message:        [1, 1, 0, 0, 1, 0, 1]
Random:                  [-1, -1, 1, 1]
Encrypted message:       [20, 27, 25, 25, 2, 4, 8, 0, 8, 31, 14]---- 101 ----
Sender's Message:        [1, 1, 0, 0, 1, 0, 1]
```



```
enc [[20, 26, 26, 25, 1, 4, 7, 0, 8, 31, 14], [20, 27, 26, 25, 2, 4, 8, 0, 8, 31, 14], [20, 27, 25, 26, 2, 5, 7, 0, 8, 31, 14], [20,
27, 25, 25, 2, 4, 8, 0, 8, 31, 14], [20, 27, 25, 25, 2, 4, 8, 0, 8, 31, 14], [20, 27, 26, 25, 2, 4, 7, 0, 8, 31, 14], [20, 27, 26, 25
, 2, 4, 7, 0, 8, 31, 14], [20, 26, 25, 25, 1, 4, 7, 0, 8, 31, 14], [20, 26, 25, 26, 2, 5, 7, 0, 8, 31, 14], [20, 27, 25, 26, 1, 4, 8,
 0, 8, 31, 14], [20, 27, 25, 25, 1, 5, 8, 0, 8, 31, 14], [20, 27, 25, 25, 2, 4, 8, 0, 8, 31, 14], [20, 26, 25, 25, 1, 4, 7, 0, 8, 31,
 14], [20, 26, 26, 25, 2, 4, 7, 0, 8, 31, 14], [20, 27, 25, 26, 2, 5, 8, 0, 8, 31, 14], [20, 26, 25, 25, 1, 4, 7, 0, 8, 31, 14], [20,
 26, 25, 26, 2, 4, 8, 0, 8, 31, 14], [20, 27, 25, 25, 2, 4, 8, 0, 8, 31, 14], [20, 27, 25, 25, 2, 4, 8, 0, 8, 31, 14], [20, 27, 26, 2
5, 2, 4, 7, 0, 8, 31, 14], [20, 26, 25, 25, 1, 4, 7, 0, 8, 31, 14], [20, 26, 26, 26, 1, 4, 8, 0, 8, 31, 14], [20, 27, 25, 26, 2, 5, 8
, 0, 8, 31, 14], [20, 27, 26, 25, 2, 4, 8, 0, 8, 31, 14], [20, 26, 26, 25, 1, 4, 7, 0, 8, 31, 14]]
Decrypted message:       [1, 0, 1, 0, 0, 0, 0]
Decrypted message:       [1, 1, 1, 0, 1, 0, 1]
Decrypted message:       [1, 1, 0, 1, 1, 1, 0]
Decrypted message:       [1, 1, 0, 0, 1, 0, 1]
Decrypted message:       [1, 1, 0, 0, 1, 0, 1]
Decrypted message:       [1, 1, 1, 0, 1, 0, 0]
Decrypted message:       [1, 1, 1, 0, 1, 0]
Decrypted message:       [1, 0, 0, 0, 0, 0]
Decrypted message:       [1, 0, 0, 1, 1, 1, 0]
Decrypted message:       [1, 1, 0, 1, 0, 0, 1]
Decrypted message:       [1, 1, 0, 0, 0, 1, 1]
Decrypted message:       [1, 1, 0, 0, 1, 0, 1]
Decrypted message:       [1, 0, 0, 0, 0, 0]
```



```
dec [[1, 0, 1, 0, 0, 0, 0], [1, 1, 1, 0, 1, 0, 1], [1, 1, 0, 1, 1, 1, 0], [1, 1, 0, 0, 1, 0, 1], [1, 1, 0, 0, 1, 0, 1], [1, 1, 1, 0,
1, 0, 0], [1, 1, 1, 0, 1, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 1, 1, 1, 0], [1, 1, 0, 1, 0, 0, 1], [1, 1, 0, 0, 0, 1, 1], [1, 1, 0, 0, 1
, 0, 1], [1, 0, 0, 0, 0, 0], [1, 0, 1, 0, 1, 0, 0], [1, 1, 0, 1, 1, 1, 1], [1, 0, 0, 0, 0, 0], [1, 0, 0, 1, 1, 0, 1], [1, 1, 0, 0, 1,
 0, 1], [1, 1, 0, 0, 1, 0, 1], [1, 1, 1, 0, 1, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 1, 1, 0, 0, 1], [1, 1, 0, 1, 1, 1, 1], [1, 1, 1, 0,
1, 0, 1], [1, 0, 1, 0]]


--start--
[1, 0, 1, 0, 0, 0, 0] 80
[1, 1, 1, 0, 1, 0, 1] 117
[1, 1, 0, 1, 1, 1, 0] 110
[1, 1, 0, 0, 1, 0, 1] 101
[1, 1, 0, 0, 1, 0, 1] 101
[1, 1, 1, 0, 1, 0, 0] 116
[1, 1, 1, 0, 1, 0] 58
[1, 0, 0, 0, 0, 0] 32
[1, 0, 0, 1, 1, 1, 0] 78
[1, 1, 0, 1, 0, 0, 1] 105
[1, 1, 0, 0, 0, 1, 1] 99
[1, 1, 0, 0, 1, 0, 1] 101
[1, 0, 0, 0, 0, 0] 32
[1, 0, 1, 0, 1, 0, 0] 84
```



```
--start--
[1, 0, 1, 0, 0, 0, 0] 80
[1, 1, 1, 0, 1, 0, 1] 117
[1, 1, 0, 1, 1, 1, 0] 110
[1, 1, 0, 0, 1, 0, 1] 101
[1, 1, 0, 0, 1, 0, 1] 101
[1, 1, 1, 0, 1, 0, 0] 116
[1, 1, 1, 0, 1, 0] 58
[1, 0, 0, 0, 0, 0] 32
[1, 0, 0, 1, 1, 1, 0] 78
[1, 1, 0, 1, 0, 0, 1] 105
[1, 1, 0, 0, 0, 1, 1] 99
[1, 1, 0, 0, 1, 0, 1] 101
[1, 0, 0, 0, 0, 0] 32
[1, 0, 1, 0, 1, 0, 0] 84
[1, 1, 0, 1, 1, 1, 1] 111
[1, 0, 0, 0, 0, 0] 32
[1, 0, 0, 1, 1, 0, 1] 77
[1, 1, 0, 0, 1, 0, 1] 101
[1, 1, 0, 0, 1, 0, 1] 101
[1, 1, 1, 0, 1, 0, 0] 116
[1, 0, 0, 0, 0, 0] 32
[1, 0, 1, 1, 0, 0, 1] 89
[1, 1, 0, 1, 1, 1, 1] 111
[1, 1, 1, 0, 1, 0, 1] 117
[1, 0, 1, 0] 10



--decrypt msg

 Puneet: Nice To Meet You
```

Conclusion and Future Work:

In conclusion, a chat application based on the NTRU encryption algorithm offers users a quick and safe way to interact. As compared to more established encryption algorithms like RSA and ECC, the NTRU encryption technique offers faster both encryption and decryption speeds. Also, the NTRU is almost resistant to quantum attacks and not breakable.

The future work might concentrate on a number of areas, but there is always potential for improvement. Usability may be one of the main areas in need of development. Although the NTRU encryption technique is very effective, it is difficult to apply on devices with limited resources, such as smartphones, because it needs a lot of CPU power. Future work might concentrate on creating more effective NTRU algorithm implementations to enhance the application's usability on such devices. Although the NTRU method is quite safe, it is still crucial to keep up with the most recent security best practises and make sure the application is still shielded from new threats.

References:

[1] A Novel Cryptosystem for Files Stored in Cloud using NTRU Encryption Algorithm. (2020). International Journal of Recent Technology and Engineering Regular Issue, 9(1), 2127-2130. doi:10.35940/ijrte.a2536.059120

[2] M. E. (2016). Modifying Shor's algorithm to compute short discrete logarithms.

[3] A. P. (2007). Performance Analysis of Public key Cryptographic Systems RSA and NTRU.

[4] Gerjuoy, E. (2005). Shor's factoring algorithm and modern cryptography. An illustration of the capabilities inherent in quantum computers. American Journal of Physics, 73(6), 521-540. doi:10.1119/1.1891170

[5] Perlner, R. A., & Cooper, D. A. (2009). Quantum resistant public key cryptography. Proceedings of the 8th Symposium on Identity and Trust on the Internet - IDtrust 09. doi:10.1145/1527017.1527028

[6] Sms Encryption Using Ntru Algorithm, Er. Amanpreet Kaur, Er. Navpreet Singh, April 2015, International Journal Of Advanced Research In Computer Science & Technology.

[7] Ekerå, M., & Håstad, J. (2017). Quantum Algorithms for Computing Short Discrete Logarithms and Factoring RSA Integers. Post-Quantum Cryptography Lecture Notes in Computer Science,347-363. doi:10.1007/978-3-319-59879-6_20

[8] Kaur, A., & Singh, N. (2015). SMS Encryption using NTRU Algorithm. International Journal of Advanced Research in Computer Science & Technology.

[9] Duits, I. (2019). The Post-Quantum Signal Protocol Secure Chat in a Quantum World.

[10] Gaithuru, J. N., & Bakhtiari, M. (2014). Insight into the operation of NTRU and a comparative study of NTRU, RSA and ECC public key cryptosystems. 2014 8th. Malaysian Software Engineering Conference (MySEC). doi:10.1109/mysec.2014.6986028

[11] Hermans, J., Vercauteren, F., & Preneel, B. (2010). Speed Records for NTRU. Topics in Cryptology - CT-RSA 2010 Lecture Notes in Computer Science, 73-88. doi:10.1007/978-3-642- 11925-5_6