NTRU Python Library with Application to Encrypted Domain

Sunil Philip
Electrical Engineering Department
Columbia University
sp3174@columbia.edu

Abdus Samad Khan
Industrial Engineering and Operations Research
Columbia University
ak3674@columbia.edu

Abstract—**An important problem in data analytics is how to collect user data and make it available for analysis without compromising user privacy. A recommended solution is to encrypt data using ring homomorphism methods which allows analysis to be performed on encrypted data, allowing the analyst to decrypt the calculation without knowing the individual private data elements used in the analysis. Lin, Shieh and Wu [2] showed that the existing NTRU encryption system could be modified to provide just this level of functionality, but open source tools do not currently exist to support NTRU. Our ntru.py and poly.py libraries are designed to perform the operations needed to broadly implement NTRU encryption as well as the application of NTRUEncrypt for domain encryption.**

Keywords-NTRU; lattice; cryptography; encrypted domain; python library

I. INTRODUCTION

Motivation: A tremendous utility in the age of big data is the collection and sharing of data for the common good. Specifically users share their information to central repositories and this allows other users and third parties to use this shared information for analysis. While there is a great deal of benefit to shared data, a major issue is the protection of the user's privacy while still allowing other users the benefit of performing analysis.

Solution: So what is a viable solution? Research by Shieh, Lin, and Wu proposes the creation of a recommendation system with an end-to-end encrypted domain. In fact, to make such an encryption scheme useful requires ring homomorphisms for the encryption, because to effectively perform calculations in our encrypted domain two operations are needed: multiplication and addition. Specifically, the encryption scheme is based on lattice-based polynomial ring homomorphism and is a special case of the NTRU encryption scheme. This NTRU based method is an improvement on RSA type encryption methods, which only use one operation, and as an added benefit, NTRU appears to be resistant to quantum computer based.

Our Contribution: In spite of the obvious advantages of NTRU encryption, there is a lack of freely available software packages for the implementation of lattice based encryption. Our project is the creation of a python library of functions which implement the algorithms necessary for an NTRU encryption of polynomials as described below. Specifically, for the purposes of our project we will also test the implementation of the libraries functions for recommendation domain encryption methodology described by JR Shieh, Lin, and Wu to test whether single messages and full scores can be encrypted and decrypted without noise.

II. RELATED WORKS

As mentioned earlier, the software package implements necessary tools to encrypt and decrypt message with lattice based polynomial homomorphisms. Paid packages, such as Maple, exist which implement these algorithms. Also our sample calculations, will implement a specific case of NTRU described by Shieh, Lin, and Wu for the implementation of a lattice based encrypted domain for recommendation systems. Specifically we will show that a user score which is a combination of multiplications and summations can be calculated, encrypted and decrypted and the original score can be recovered [2].

III. SYSTEM OVERVIEW

NTRU encryption is an Asymmetric Cryptography System (also known as Public-Key Cryptography), therefore in-order to understand NTRU it is important to first understand how an asymmetric cryptography works.

Consider the case where Alice wants to send a private message to Bob using the NTRU encryption system. As in the case of all Public-key cryptography systems Bob is required to generate a pair of keys: public and private. Public key as its name suggests will be available to everyone. This key will be used to encrypt messages, which will be decodable using the private key only Bob will possess.

In the NTRU encryption system all objects including messages and keys are elements of a polynomial ring. If the reader comes across any unfamiliar notions please reference the Appendix. We will discuss the algorithm below for encrypting and decrypting using NTRU and the NTRU variation

IV. ALGORITHM

We will give a brief over view of the NTRU encryption in this section. Our focus will be on the steps required to

successfully encrypt and decrypt using this, covering only the important steps and skipping proofs.
As mentioned in this method we work with polynomials with integer coefficients. Let $\mathbb{Z}[x]$ denote the set of all polynomials with coefficients in $\mathbb{Z}$. Prior to use, NTRU requires integer parameters (N, p, q) to be specified. Note that NTRUEncrypt is a special case of NTRU and actually requires the passing of four integer parameters (N, p, q, d). These parameters are publically known and must meet certain conditions.
N: all polynomials will have degree strictly less than this number and N must be prime.
p, q: parameters publically known and are used in encryption and decryption.

1. They must be co-prime.
$$\gcd(p, q) = \gcd(N, q) = 1$$
2. $q > p$
3. $q > (6d + 1)p$ (NTRU Encrypt)

Condition (3) is the needed to implement the variation of NTRU called NTRUEncrypt and is used to implement the inner product encryption suggested in [2] for recommendation systems.

Key Generation.
The key maker (Bob in the previous section) will have to generate the public key and private key pair. He will specify two functions f, g along with the parameters (N, p, q). These functions are generally chosen randomly but must satisfy the following properties.

- Must have order strictly less than N.
- Must have coefficients belonging from {1,0,-1}
- f must be such that its modulo q and modulo p inverses must exist, i.e there exists f_p, f_q such that $f_p f = 1\ (mod\ p)$and $f_q f = 1\ mod\ q$

For NTRUEncrypt we specify that $f \in T(d + 1, d)$ while $g \in T(d, d)$ (See Appendix for definition)

Generate Public Key and Encryption:

1) Find modulo p and modulo q inverses of f denoted by f_p and f_q using the Extended Euclidean Algorithm.
2) Then $h = \left[f_q g\ mod(x^N - 1)\right] mod\ q$ gives the public key

To encrypt message using public key we first convert message into a polynomial with binary coefficients. We call this polynomial m. Along with this we also generate a random ternary polynomial r to add noise to our encrypted message. Then the encrypted message is calculated as

$$e = prh + m\ mod(x^N - 1)] mod\ q \qquad \text{(Eq 1)}$$

Now this message can be sent to the key maker (Bob).

Decryption NTRU:
We apply the following function to the encrypted message e
$$[f \cdot e\ mod\ (x^N - 1)]\ mod\ p \qquad \text{(Eq 2)}$$

Next perform centered lift on the resulting polynomial using q (please see Appendix).
Finally we perform the operation following calculation
$$[f_p \cdot e\ mod\ (x^N - 1)]\ mod\ p \qquad \text{(Eq 3)}$$
Now the result of (Eq 3) is m.

Encryption and Decryption NTRUEncrypt for Recommendation Systems:

The subsection Decryption NTRU describes the general application of the NTRU library as a utility for users. But we also wish to show the modified use of NTRU as it applies to a specific big data problem. The authors of [2] showed that if we wish to encrypt a score calculation (using multiplication and addition) and send it to Bob. authors use separate binding polynomials r_i each for a different message m_i and apply (Eq 1) to encrypt

$$e_i = [pr_i h + m_i\ mod(x^N - 1)] mod\ q \qquad \text{(Eq 1)'}$$

So we want Bob to be able to perform analysis, say that he needs to be able to multiply and add the m_i's to come up with a score, but Alice (the sender) does not want him to know the individual e_i's. She will send Bob
$$e = e_1 \cdot e_2 + \cdots + e_{n-1} \cdot e_n$$

Bob will then apply the following modifications to NTRUEncrypt's Decryption

$$[f^2 \cdot e\ mod\ (x^N - 1)]\ mod\ p \qquad \text{(Eq 2)'}$$

Apply centered lift to the result and then apply
$$[f_p^2 \cdot e\ mod\ (x^N - 1)]\ mod\ p \qquad \text{(Eq 3)'}$$
After (Eq 3)' the result will be
$$m_1 \cdot m_2 + \cdots + m_{n-1} \cdot m_n$$

So if Bob needs to perform analysis which requires the ring operations he can do so and he will not need to know the individual data elements m_i.

A number of other algorithms were also implemented to in this software library. The function names are given in Section V of this report and detailed description are given in the documentation found in the repository.

V. Software Package Description

We have built a python package which provides an implementation of NTRU Encryption System. To ensure

accuracy of the encryption and decryption we required to know polynomials with high accuracy. Since existing polynomial libraries such as the one provided by the Numpy Python Package has coefficients as floats, we could not employ them. Therefore, we implemented a fresh polynomial package which allows us to perform operations on polynomial with rational coefficients. A brief description of both packages is provided below. For detailed description please refer to the documentation provided in our repository.

Ntru.py (Documentation: NTRU.md): Contains definition of Ntru class object

genPublicKey(f,g,d): Generates a public key

setPublicKey(public_key): Sets class-variable h (public key) to the given custom public_key

getPublicKey(): getter function for class variable h (public key)

encrypt(m,randPol): Encrypts given message m using a random polynomial randPol and public key. (Note that before calling this function you need to either set the public key or generate it)

decrypt(en): This method decrypts the given message using private key information stored during the generation of the public key. Therefore can only be used once the public key has been generated.

decryptSQ(e): Decrypts messages using a slightly different approach used for analytics in encrypted domain (refer to report)

poly.py (Documentation: POLY.md)

This library allows the user do mathematical operations on rational coefficient polynomials. For rational coefficients we have used fractions data type which is a standard library in Python.

addPoly(c1,c2): Returns addition of polynomials c1 and c2
subPoly(c1,c2): Returns subtraction of c2 from c1
*multPoly(c1,c2):*Returns product of c1 and c2
divPoly(c1,c2): Returns the quotient and remainder of c1/c2
cenPoly(c1,q): Returns the centered lift of the given polynomial
resize(c1,c2): Adds leading zeros to the smaller of the two vectors which represent polynomials.
trim(c1): Removes Leading zeros from the input vector representing the polynomial
modPoly(c1,k): Returns a polynomial with the coefficients of c1 modulo an integer k.
*isTernary(f,alpha,beta):*Checks if the polynomial is a Ternary polynomial returns a Boolean value
extEuclidPoly(a,b) : Returns [gcd(a,b),s,t] where s and t are Bezout polynomials.

* All functions above work with fraction coefficients

VI. Experiment Results

To test the accuracy of the library, we used the same inputs for encryption and decryption used by Dr. Shieh. We compared the results of each function in our library with an identical calculation performed by Dr. Shieh in Maple and successfully replicated his results.

VII. Conclusion

The conclusions about the methods discussed and implemented have already been peer-reviewed. We achieved our objective to provide accurate open-source libraries for the public to implement these methods. The contributions of our team members are as follows:

- Read Papers on Ring Homomorphism Encrypted Domain and Deconstruct Necessary Algorithms – S Philip 80%, AS Khan 20%
- Implement/Code algorithms from Step 1 – S Philip 30%, AS Khan 70%
- Write Technical Report and Screencast – S Philip 50% , AS Khan 50%

Acknowledgment

The authors would like to thank **Jye-Ren Shieh** and . **Ching-Yung Lin** for their support and direction in the area of encrypted domain data mining techniques.

Appendix

Mathematical Notation and Concepts

Let R be any ring (e.g. $\mathbb{Z}$ =the integers) then
$R[x] = set\ of\ all\ polynomials\ with\ coefficients\ in\ R$

$\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ =Equivalence classes of integers which have the same remainder when divided by p.

$R[x]/(g(x))$. Let $g(x)$ be a polynomial in $R[x]$, then $R[x]/(g(x))$ =set of polynomial in $R[x]$ which are considered equivalent if when divided by $g(x)$ they have the same remainder.

$T(d_1, d_2)$= collection of ternary polynomials (polynomials with coefficients in {-1,0,1}) such that each polynomial in

our collection has the coefficient 1 appearing times d_1 and the coefficient -1 appearing d_2 times.\

Centered Lift: Let $(x) \in \mathbb{Z}_q[x]\ ]$. The *centered lift* of $a(x)$ is the unique polynomial $a'(x) \in \mathbb{Z}[x]$ satisfying

$$a'(x) \equiv a(x)\ mod\ q$$

Whose coefficients a_i' are in the interval

$$-\frac{q}{2} < a_i' < \frac{q}{2}$$

REFERENCES

[1] J. Hoffstein, J. Pipher, and J.H. Silverman, Introduction to Mathematical Cryptography, Springer,-Verlag, New York, NY, 2008.

[2] CY Lin , JR Shieh, and JL Wu, "Recommendation in the End-to-End Encrypted Domain," in Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM '11. New York, USA: ACM Press, 2011.

[3] JR Shieh,, "Ecnrypted-Domain Data Mining for Recommendation".