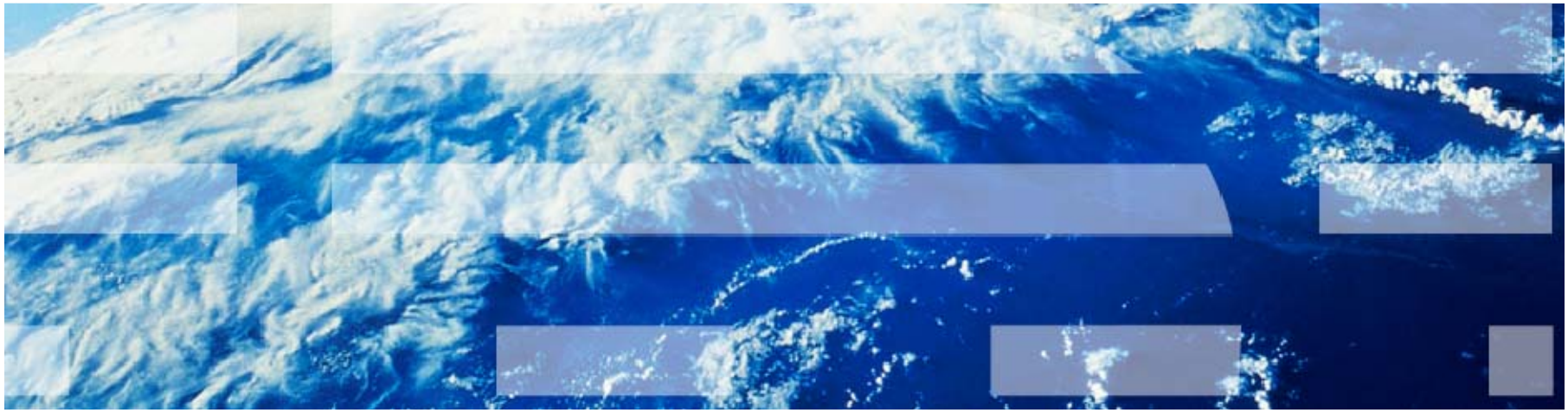
COLUMBIA UNIVERSITY

E6895 Advanced Big Data Analytics:

An Open Source Lattice Based Cryptographic Library

Team Members: Abdus Samad Khan & Sunil Philip



May 18, 2015

Motivation

A tremendous utility in the age of big data is the collection 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 the users of this data, a major issue is the protection of the user's privacy while still allowing other users the benefit of performing analysis.

What could go wrong?

Oh NO!!! Someone ROBBED Me!!!

Solution

A series of proposals has appeared in the literature which suggests lattice based ring homomorphism encryption to encrypt the domain.

Idea: We still want to be able to encrypt information that the users sends to a server and anyone using the user data on the server for analysis will only need information aggregated and encrypted by the original user.

So the user may have message $m_1 = 2, m_2 = 5, m_3 = 7, m_4 = 1$. The user recommendation is defined by $m_1 \cdot m_2 + m_3 \cdot m_4 = 2 \cdot 5 + 7 \cdot 1 = 17$. We want to encrypt $\epsilon(m_1 \cdot m_2 + m_3 \cdot m_4) = \epsilon(m_1) \cdot \epsilon(m_2) + \epsilon(m_3) \cdot \epsilon(m_4) = \epsilon(17)$. Then the host server can decrypt the aggregate score, but not the individual messages m_i, unless the user transmits them separately.

Our Contribution

COLUMBIA UNIVERSITY

- Thankfully, the mathematical framework for such an encryption system already exists, called NTRU.
- Lin, Shieh, and Wu proposed an augmentation to this system, which would encoding and recovery of the multiplication and summation of the original m_i's without compromising the original m_i's.
- We realized that the algorithms for NTRU are only available on paid platforms, such as Maple.

✓ Our project is a Python library that implements all of the fundamental algorithms for NTRU encryption and decryption.

Dataset, Algorithm, and Tools

COLUMBIA UNIVERSITY

Dataset:

- We have a series of sample calculations, previously published by Jye-Ren Shieh, implementing a variation of NTRU. We will also run other polynomials through the NTRU algorithms implemented in Maple.

Algorithms (All of the algorithms defined below only accept and return rational coefficients):

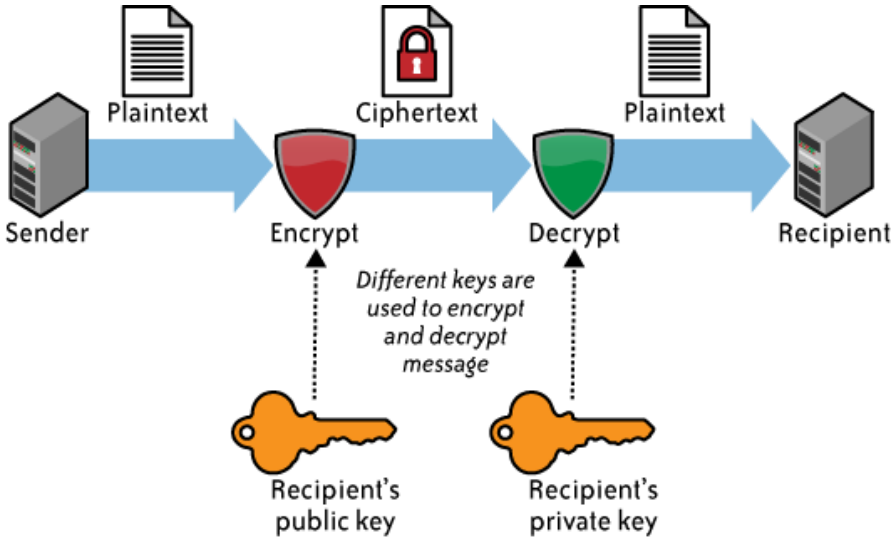
- Polynomial Addition, Subtraction, Multiplication, and Division
- Extended Euclidean Algorithm for Polynomial
- Extended Euclidean Algorithm for Integers
- Modular Inversion
- NTRU Public Key generation
- NTRU Encryption and Decryption

Tools:

- Python – creation of library of algorithms above
- Maple – to test accuracy of algorithm implementation in Python

NTRU: An Asymmetric Encryption System

- Two separate keys generated by the receiver of the secure message
- Public key used for encryption
- Private key is used for decryption



- NTRU works on this framework
- All objects are polynomials

Python Library

COLUMBIA UNIVERSITY

- 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.
- Therefore, we implemented polynomial package which allows us to perform operations on polynomial with rational coefficients.

poly.py

ntru.py

Polynomial Library poly.py

Columbia University

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

- *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.

NTRU Library ntru.py

Contains definition of Ntru class object and methods

- *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(e)*: 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)

Application in Encrypted Domain

- To test our library as a tool for encrypted domain we followed an example presented in a paper. Using our library we were successfully able to reproduce the results

Decryption

$$\alpha \equiv (\boldsymbol{f}(\boldsymbol{x})^2(e_1^a(x)e_1^k(x) + e_2^a(x)e_2^k(x)+e_3^a(x)e_3^a(x)))mod\ \mathrm{R}_q[x]$$

$$= 468645x^6 + 475249x^5 + 1531x^4 + 18383x^3 + 23015x^2$$

$$+ 8623x + 479159$$

Centered lift coefficients

$$\alpha = -22886x^6 - 16282x^5 + 1531x^4 + 18383x^3 + 23015x^2 + 8623x - 12372$$

$$\boldsymbol{F_p(x)}^2((-22886x^6 - 16282x^5 + 1531x^4 + 18383x^3 + 23015x^2 + 8623x -$$

$$12372)\ mod\ \mathrm{R}_q[x])\ \boldsymbol{mod\ p} \quad = x^4 + 3x^3 + 3x^2 + 4x + 1$$

We then recover the **exact** plaintext similarity measurement results.

$$\boldsymbol{x^4 + 3x^3 + 3x^2 + 4x + 1 = 1 \times 2^4 + 3 \times 2^3 + 3 \times 2^2 + 4 \times 2^1 + 1 = 61}$$

In the general case $\sum_{i=1}^{n}(e_i^a(x)e_i^k(x))\ mod\ \mathrm{R}_q[x]$, the criteria of $\frac{q}{2n} > Q_n$, $Q_n = 8p^2d^3 + \frac{4}{n}p^2d^2(2d+1) + \frac{p}{2n}(2d+1)^2$ must be held. The proof of the general criteria can be found in Appendix.

We have thus demonstrated the complete procedure of ring homomorphic

22

```
akhan@drk:~/Desktop/attachments/Working Copy$ python example_enDom.py
Public Key           : [394609, 27692, 62307, 263073, 346149, 41538, 339225]
Encrypted Computation:  [56071493501, 184282540630, 461597778108, 711030608511
49, 513649362225, 314519445619, 201999272846]
Decrypted Computation:  [1, 4, 3, 3, 1]
Results match with those in the paper
akhan@drk:~/Desktop/attachments/Working Copy$
```

Basic Functionality: An Example

COLUMBIA UNIVERSITY

1) Bob is expecting to receive some secure information from Alice

2) Bob creates an instance of NTRU of with parameters (N=7,p=29,q=491531) which he makes publically available

```
>>> Bob=Ntru(7,29,491531)
```

3) Next he generates a public_key by specifying a function f,g and parameter d

```
>>> f=[1,1,-1,0,-1,1]
>>> g=[-1,0,1,1,0,0,-1]
>>> d=2
>>> Bob.genPublicKey(f,g,2)
>>> pub_key=Bob.getPublicKey()
```

Basic Functionality: An Example

COLUMBIA UNIVERSITY

4) Alice wants to send a secure message to Bob. She sets up another instance of Ntru using Bob's original parameters and public key Bob provided

```
>>> Alice=Ntru(7,29,491531)
>>> Alice.setPublicKey(pub_key)
```

5) She encrypts her message using her instance and a random Ternary polynomial for noise

```
>>> msg=[1,0,1]
>>> ranPol=[-1,-1,1,1]
>>> encrypt_msg=Alice.encrypt(msg,[-1,-1,1,1])
```

6) Finally Bob decrypts message sent to him

```
>>> print Bob.decrypt(encrypt_msg)
```

Demo