Quantum Secure Encryption Hardware Implementation of Ring Learning With Errors Senior Project 2019

Team members: Anthony Fortner, Lahiru Perera, Vasanth Sadhasivan
Faculty Advisers: Andrew Danowitz, Paul Hummel

June 20, 2019

Contents

1 Introduction **2**
1.1 Project Background 2
1.2 Potential Beneficiaries 2

2 Research & Planning **2**
2.1 LWE Basics 2
2.1.1 Private Key Generation 3
2.1.2 Public Key Generation 3
2.1.3 Encryption 3
2.1.4 Decryption 3
2.2 Why Ring LWE 4

3 Prototype design **4**
3.1 Primary Components 4
3.1.1 Gaussian Number Generator 4
3.1.2 NTT 4
3.1.3 Control Unit 4
3.2 Core Functions 4
3.3 Additional Features 5
3.4 Challenges & Revisions 5
3.5 System Diagram 5

4 Conclusions **5**
4.1 Summary of Development 5
4.2 Remaining Work & Future Development 6

1 Introduction

1.1 Project Background

As the world becomes more dependent on computer technologies, the need to secure our data becomes an ever-growing concern. With the advent of quantum computers becoming an increasing reality, we need to rethink fundamentally how we popularly handle public key encryption. There are many different approaches to handle this problem, but very few if any have architected a specific hardware acceleration system. The fundamental algorithm explored by our team, Learning With Errors (LWE), is different in the sense that it doesn't rely on the discrete logarithm problem to derive its hardness, which is known to be solvable in polynomial time with a quantum computer using methods like Shor's algorithm [1]. The LWE algorithm is based on lattice cryptography and derives its hardness from the shortest vector problem [2]. Unfortunately LWE as a base algorithm is too computationally expensive to be a feasible replacement for current encryption models and is more of a proof of concept for how data may be encrypted with lattices. To achieve a more computationally robust system rLWE is used, which ultimately utilizes a much smaller public key, and features potential optimizations with polynomial multiplication.

1.2 Potential Beneficiaries

With the ever growing need to secure our data, the possible applications for LWE based encryption are countless. The future of computing is widely thought to become dominated by quantum computers, and LWE based encryption has the potential to become our primary means to secure data. Having a secure encryption method that is resilient to the computational complexity of quantum computers eliminates one of the significant concerns of this new technology. There is also potential risk that even without quantum computers now, large amounts of network traffic thought to be secure with RSA encryption could be stored in large data centers and decrypted when quantum computer q-bit count goes up enabling mass security compromises. Therefore, a form of encryption resilient against quantum computers needs to be practically implemented as soon as possible.

2 Research & Planning

2.1 LWE Basics

Regev initially proved the hardness of the shortest vector problem and created a basic scheme for public key encryption Regev:2005:LLE:1060590.1060603.

2.1.1 Private Key Generation

To generate the private key Alice simply needs to come up with a vector that is in the vector space of n integers modulo q or Z_q^n

$$s \in Z_q^n$$

The vector elements are chosen uniformly at random $q \geq 2$, is a prime number between n^2 and $2n^2$

2.1.2 Public Key Generation

Initially Alice choose m vectors from Z_q^n uniformly and independently, let this set of vectors be set A

$$A = a_1, a_2 \ldots a_m \in Z_q^n$$

Then she chooses a random set of m errors

$$E = e_1, e_2 \ldots e_m \in Z_q^n$$

Error values are selected according to the distribution χ

$$m = (1 + y)(n + 1)log(q)$$

for an arbitrary y χ is essentially a gausian distibution over the set Z_q centered around 0 and q

We then need to compute the inner product of A with our private key S, divide by q, and add our error

$$B = \frac{< A, s >}{q} + E$$

2.1.3 Encryption

If Bob wants to give Alice a message he first needs to select a subset S of the m elements from the sets (A, B) His message is going to be a bit $x \in (0, 1)$ To encode his message he simply does this

$$(a, b) \sum_{n=1} a_i, \sum_{n=1} b_i + \frac{x}{2}$$

2.1.4 Decryption

For Alice to decrypt the set she has been given she computes:

$$b - < a, s > /q$$

If it's closer to 0 than $\frac{1}{2}$ it is a 0 Else it's a 1

2.2 Why Ring LWE

The primary reason we chose ring LWE (also known as LWE-Polynomial) versus the common alternative matrix LWE was due to speed, the size of the public and private key, and the ease of encrypiting a multibit message encoding [3]. Looking at tests performed in other papers, we saw that the matrix-based approach could be twice as slow compared to ring LWE. Secondly, we determined that polynomial based computations would be theoretically simpler to implement in hardware. Section 3 of the paper titled, On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes, has an excellent analysis of the two methods and was one of the primary resources we reference throughout the planning phase of the project.

3 Prototype design

3.1 Primary Components

3.1.1 Gaussian Number Generator

This component uses a selective reject algorithm to generate pseudo-random numbers that follow the desired Gaussian distribution. The decision to use a selective reject algorithm was due to the simplicity and speed of number generation. In our current implementation, there are still optimizations to be made to improve the speed of generation.

3.1.2 NTT

The Discrete Fourier transform or the number theoretic transform (NTT) is a critical component used in order to speed up polynomial multiplication. This is one of the key benefits of rLWE as opposed to LWE, rLWE operates in a vector space which allows the NTT operation to be applied to the lattices (in the form of polynomials), then be convolved and then go through a reverse NTT operation to get the result.

3.1.3 Control Unit

The control unit is the component that enables and disable various subsystem depending on the state of the system. Many of the systems are designed to be controlled asynchronously such as the loading of Gaussian random numbers into memory. Therefore, the control unit's functionality is critical in making sure that the needed tasks finish in the correct order and timing.

3.2 Core Functions

Our hardware implementation consists of three main functions, key generation, encryption, and decryption. Once the user has connected our hardware implementation, they can use the UART interface to send an operation code, message, and key (if needed) to the device. Key generation requires a set of Gaussian random numbers to be generated which are used to develop the key. For encryption, the device first generates a set of Gaussians random numbers then uses those values in addition to the key to performing the encryption of the message. Once the message is encrypted, the value is returned via the UART interface to be viewed by the user. Decryption happens similarly; however, the notable difference is that only the key is needed to decode the message.

3.3 Additional Features

For the ease of debugging and usability, we have implemented a UART interface. The UART interface was initially chosen to enable the simple input of messages to be encrypted and decrypted by our hardware implementation. However, while implementing the subsystems, we found that the UART interface would be useful for debugging purposes as well. The UART interface enables us to be able to quickly see the data going in and out of each component in the overall system, thus allowing for easy pinpointing of issues.

3.4 Challenges & Revisions

Our primary challenges came from the design of our three primary components: number theoretic transform (NTT), Gaussian Generator, and Control Unit. Primarily we had a difficult time coming up with the best method of linking the various components together. This indecision resulted in many revisions of the Control Unit as we originally started with a design that did not include one to having the control unit handle the various states that the system. Secondly, we went through various methods of generating and storing the Gaussian numbers. Initially, we had planned to generate these numbers in real time but quickly realized that this would cause significant delays due to our implementation of the selective reject algorithm. The primary source of the less than ideal generation time was due to the unoptimized implementation of the selective reject algorithm that we had written. We ultimately decided to make the number generation run independently from the rest of the process when the control unit raises a signal the number generation begins, and the resulting values get stored to a reserved chunk of memory.

3.5 System Diagram

4 Conclusions

4.1 Summary of Development

The majority of our time was initially spent researching about LWE and the various components that we would need to implement to successfully encrypt and decrypt a simple message. This initial step ended up taking much more time than we expected due to the complexity of the topics that was required to understand how and why the algorithm. Once we had felt we had a good understanding of the material, we began coding an implementation in Python using the NumPy library, which included many prewritten functions for the major components such as Gaussian number generation. Out implementation allowed us to confirm that our understanding of LWE was correct and move on.

With our successful Python implementation in hand, we began to design our initial system design and determine what we components we would have to develop. After a bit of discussion, we decided that we would design all the required components ourselves; this was a task that we again underestimated. Our reasoning for this choice was that we wanted to make our project's completely open and flexible to future changes by future students. Currently, we have our initial system architecture completed as well as the various components. We have begun assembling the components and have done some rudimentary testing.

4.2 Remaining Work & Future Development

The tasks that remain are mainly debugging and cleaning up of our implementation. With the various changes that we had to implement throughout the course of our project, we anticipate various bugs once all the components are assembled. Once the system is assembled and base functionality is tested, we expect that the next step will be performing an analysis of the characteristics of the device such as power consumption, speed of computations, etc. Finally, with that information in hand, the system can be optimized and improved.

References

[1] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede, “Efficient software implementation of ring-lwe encryption.” Cryptology ePrint Archive, Report 2014/725, 2014. `https://eprint.iacr.org/2014/725`.

[2] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” in *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC ’05, (New York, NY, USA), pp. 84–93, ACM, 2005.

[3] V. Lyubashevsky, C. Peikert, and O. Regev, “On ideal lattices and learning with errors over rings.” Cryptology ePrint Archive, Report 2012/230, 2012. `https://eprint.iacr.org/2012/230`.