Advanced Topics in Security and Privacy (WMCS001-05) Fully Homomorphic Encryption Project Report

Rishabh Sawhney - S4133366
Hindrik Stegenga - S3860205
Andris Jakubovskis - S3832023

October 29, 2020

1 Short introduction to FHE

Fully Homomorphic Encryption (FHE)[3] is a field of encryption techniques where the primary idea is that computations can be performed directly on the ciphertext .The result is obtained in the form of an encrypted ciphertext which can then be decrypted using the public key as well as the secret key. There are different levels of Homomorphic Encryption namely:

1. Partial Homomorphic Encryption
2. Somewhat Homomorphic Encryption
3. Leveled Homomorphic Encryption
4. Fully Homomorphic Encryption

These different types of encryption are ideally represented as Boolean or arithmetic circuits. These different types of encryption usually differ in the types of operations and the different circuits they can act on. Fully Homomorphic encryption is able to perform operations on arbitrary circuits of unbounded depth and is the strongest type of encryption among the other Homomorphic Encryption schemes. Homomorphic Encryption schemes are usually malleable meaning they will offer lesser security over non-homomorphic encryption schemes.

A basic schematic for the working of **Fully Homomorphic Encryption(FHE)** toolkit can be viewed in Figure 1 .Here, an operation f is to be performed over the encrypted value of m given by $Enc(m)$ such that the encrypted value of $f(m)$ given by $Enc(f(m))$ will give the same value as $f(m)$.

Scope of the project : The scope of our project was to perform FHE operations using the IBM FHE Toolkit developed by IBM, which is available either as a linux docker image or the macOS variant which uses Xcode. It primarily consists of **HELib**[4]. This library is primarily built in C++. For this project, we have performed operations composed of addition and multiplication using the different encryption schemes namely, **Brakerski-Gentry-Vaikuntanathan (BGV)**[1] and **Cheon-Kim-Kim-Song (CKKS)**[2] to some complex operations such as obtaining 2x2 as well as 3x3 matrix determinants as well as multiplication of 3x3 matrices with complex numbers along with other operations of varying complexity.

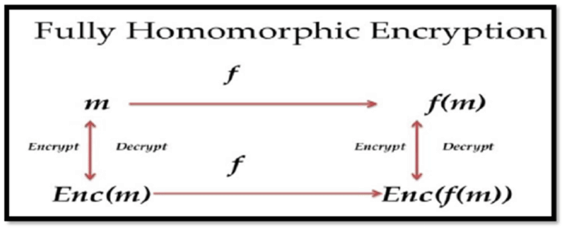

Figure 1: Schematic representation for FHE[3]

Applicability of FHE in industry : Homomorphic Encryption presents the method to perform computations on a server which cannot know or be able to decrypt the contents or results of the computations by itself without the user's intervention. This can be useful for performing privacy preservation based operations such as storage or computation on encrypted data on the server. This can be applied to various fields of industry such as medicine,finance where privacy is of utmost importance. Operating solely on encrypted data eliminates majority of the privacy based concerns in these industries.

2 Experiences, Discussion and Reflection

2.1 Hindrik

For me the primary hurdle in getting started on the project was the theoretical complexity involved in the used algorithms. The IBM-FHE toolkit is primarily built around HELib, which is a library intended for security researchers. This shows very much in it's design, it is very unforgiving if you don't understand what each parameter does and how they affect the FHE process. HELib implements both BGV and CKKS schemes, and there are many ways in which you can use those in HELib. We had a lot of trouble getting even the simplest things working for a long time due to this. It didn't help that the examples provided by both the FHE-Toolkit and HELib are not that straightforward to extract the workings from.

In order to solve this situation, we put a lot of effort into reading various papers and articles about both schemes, but the required mathematics to fully understand them is pretty much out of reach for us given the time frame and scope of the project. However, they definitely did help in understanding how the current FHE schemes work. I got a good idea about how the Shortest Vector Problem, the basis of all current FHE schemes (Although in different forms like LWE and RLWE), works and therefore why the addition of (controlled) noise eventually causes decryption errors. It also gave me a good insight in why we need bootstrapping in order to go from a limited amount of operations to any amount of operations. Although practically you're still somewhat limited by the computational and memory constraints because of this bootstrapping process. Effectively, bootstrapping increases the total noise allowable in the system by re-encrypting the previous result with a new key when a certain threshold of noise has been reached. This way, it turns a so-called 'Somewhat Homomorphic Scheme' into a fully homomorphic scheme. Although practically you will still be limited since bootstrapping causes a massive increase in the required computations as well as required memory. If the amount of operations is small it might be possible to not even need bootstrapping at all.
The next tricky part I don't quite understand is modulus switching, which is used for both BGV and in a different form for CKKS. From my basic understanding it reduces the noise introduced after a multiplication from exponential to linear. While I had no time to figure out precisely how this works, nor how CKKS precisely works, I managed to get some basic examples working. Atleast from a high level I understand what CKKS allows you to do: compute on complex and fixed point numbers with some approximation.

After all this, I started working on implementing various examples using both BGV and CKKS encryption schemes. This definitely helped me in understanding certain parameters and how they relate to each other as well. I also learned that apparently you can use polynomials to do computations, something I had never heard or seen before. Once I got the basic operations working I finally understood how to practically use HELib in order to perform computations on ciphertexts. Once you have setup all the required parameters, like the prime, modulus and a few others, the basic operations are quite straightforward to implement. Effectively you apply the as you normally would on numbers, but now you apply them to the ciphertexts instead. I also found that HELib notifies you when a potential decryption failure occurred, which is quite handy. This indicates that you perform too many operations causing the noise threshold to be exceeded, which meant I had to either apply bootstrapping or change the used prime number and modulus chain depth. While I might not fully understand the mathematics involved in setting these parameters, at least I somewhat got an idea on what to do whenever I need to change them.

Another interesting restriction I found out when trying to implement Euler's Algorithm to compute the GCD of two numbers is that certain algorithms are practically not implementable in FHE schemes. While you might be able to do any computational algorithm using multiplications and additions, condition based logic is not always possible. Since the logic requires you to evaluate the (plain text) result, which you don't have, you can't include if statements for example. In other words, you can't perform an if check on cipher texts and expect it to work. In certain cases you might be able to work around it changing the computation such that it will multiply with 0 or 1 corresponding to true false, but that's not always possible or practical. Similar problems occur when you want to use algorithms requiring recursion. If you have a large possible recursion depth, this requires you to allow for a potentially large amount of computations, thus leading to impractically large bootstrapping sizes and costs. Another problematic downside of FHE is that you're restricted in what operations you can perform. This is something I encountered when trying to implement finding the GCD. For example, division is not possible in most cases, and also operations like mod and square root are not available. Therefore, the only other option I would have would be to use the binary GCD algorithm, but sadly right bit shifts are required but that's not an available operation. This means that certain algorithms cannot be implemented, or at least not efficiently. Since computing the GCD requires both recursion and conditional logic, as well as either mod or division or right-shift depending on the specific algorithm you use, I didn't make any further time-costly attempts implementing it.

Our choice for using the macOS version of the FHE-Toolkit worked out well for me, since I have a job as iOS developer for a couple years already. Creating applications for macOS is quite similar, with the exception that I had to use Objective-C++ rather than Swift, and that certain APIs are quite different as compared to their iOS counterparts. As such, I took care of most of the UI work for the project.

So to conclude, while I initially had a lot of problems getting started with the project, eventually I started to understand the different parts involved in perform FHE computations. If I were to do such a project again, I would probably look into different libraries which are more accessible for a beginner in FHE to use, as HELib is not very well suited for that.

2.2 Rishabh

There were several difficulties in understanding the different aspects of FHE namely,the different types of Encryption schemes and the different components of the toolkit in general which took a fair about of time to understand.

Initially we were in the frame of mind of understanding the basic theory behind all the various elements that were a part of Fully Homomorphic Encryption such as Ring Learning With Errors(RWLE) as well as some other concepts such as Lattice Theory and spent many days working on trying to understand at least the basics, to no avail.The HELib toolkit primarily works as a Docker image with an IDE as well a part of macOS through Xcode. This was however, an entirely different experience of it as I hadn't used it before. These included making out the storyboards for the UI as well as programming in Objective-C++ which were entirely new to me at this point. On trying the build the toolkit,my Xcode was causing errors as it was not able to find certain libraries which were working fine for Hindrik and Andris. Thus I was working on the Linux toolkit which was based on the docker image.

Once I somewhat understood how the toolkit works with the basic algorithms as example and took me a lot of time to properly understand the different elements in it such as encrypting and the use of vectors which I didn't have much experience in but got appropriate help and support from my group members which was a good thing.

The main issue was implementing something novel that was based on the examples that were being worked on,namely Matrix multiplication which required me to encrypt the individual elements in the vector and then store the values in a cipher text matrices for both the matrices that were to be multiplied. This was done with a lot of help from Hindrik and have since we have got it working.

I was able to figure out the basic operations such as addition and multiplication of two integers along with performing scalar product of two vectors.However, these have also been successfully implemented by my group members.I was able to perform complex multiplication which basically multiplied the corresponding real and the imaginary parts. This took a bit of time to understand as I had previously not worked on complex numbers in C++ before. Post this, I also worked on performing the rotation operation on the complex variable. However, it was observed that there were some inconsistencies regarding the output for the decrypted values after rotation for large complex numbers.

Although, FHE helps perform operation on encrypted data, it still has certain limitations on the type of operations that can be performed under this scheme. Also, operations on cipher text take considerably longer to be implemented due to the encryption overhead involved. Me and my team mates were unable to narrow down the different parameters involved in reducing this computational overhead.

I was however, able to grasp the various concepts introduced in Fully Homomorphic Encryption at least in theory. The toolkit even though extensive still lacks some kind of instruction manual which can be used by beginners to play around with the different operations in the toolkit. This would be helpful for getting different people getting their hands of the tool kit as it is pretty powerful and can lead to further progress in the field of Homomorphic Encryption

2.3 Andris

The first thing when I started working on the project did was setting up FHE-toolkit. We noticed that there are two implementations offered by IBM on GitHub - Linux and macOS. We opted for macOS version since it was the operating system on all of our machines. The instructions on installing the toolkit were clear and I managed to build the example projects that were included in the toolkit. The examples included, however, did not really capture what we were trying to achieve with the project. After researching the internet for more information on how to execute mathematical operations with FHE, I discovered that the information available on the internet on this topic is quite limited. Moreover, the resources to be found on the internet involved complex math concepts such as lattices , BGV encryption, ring-lwe. After spending few days on trying to understand these concepts and how do they relate to the project I did not manage to understand them. At this moment I was quite frustrated similarly as other group members. Yet, we decided on trying to implement the examples without completely understanding how the encryption and decryption was working.

Our project is implemented by the use of Objective-C and C++. This was my first experience working with Objective-C. I had small prior experience with Swift, despite that Objective-C proved to be tougher to comprehend. Therefore, I spent some time learning about how to implement the user interface with Objective-C. I found it very convenient that one can write C++ code in Objective-C file. After learning how to use the UI components in the code by referencing outlets, I worked on implementing the quadratic polynomial calculation. It took me a while to create this example. During my work on it I noticed that the lack of theory knowledge was slowing me down but by looking at examples on understanding the implementation, I was able to implement it. Initially, I was getting the wrong result back. However, in the output the program informed me that there might be some problem with decryption. By changing the bit parameter values, I eventually was able to obtain an accurate result. When the example was finished I noticed that the layout I had made in storyboard is significantly different from the one that is being shown when the app in being running.

When creating the complex conjugate example, I luckily did not have this issue anymore. However, initially I was planning to implement it by using CKKS encryption. Later on I decided to go for BGV encryption instead since there were some problems with not matching data types and therefore I was not able to execute the `conj()` function in HeLib library. The implementation of this example was way smoother as I already understood better how to use the toolkit.

Overall, the project definitely proved to be a challenge as it felt like extensive foreknowledge of how the encryption algorithms work is required. I believe the most time of the project was spent trying to comprehend the theory on how to implement the operations on encrypted data. Currently, I wish we had went for trying to implement the examples earlier without before trying to understand the theory behind it. However, once I started writing the code project execution accelerated. Although I was able create examples for the project, I did not completely understand why my code works which was a shame. On the other hand I learned quite a lot about Homomorphic encryption, C++ and Objective-C.

3 Final remarks

In conclusion, while performing the different operations, we observed that only computations such as multiplication and addition can be applied on the cipher texts. This narrows down a lot of operations that can be performed on the encrypted data. The toolkit also does not support operations such as exponents and recursion along with division although the cipher text can be divided by 2 in some cases.The toolkit supports operations on complex as well as floating point numbers as well as support for polynomial based computations through the means of CKKS encryption scheme.

There is still research taking place in Homomorphic Encryption and hopefully the toolkit can be expanded to take the limitations addressed previously going forward.The learning curve for getting into the field is high and presently documentation for the toolkit is somewhat limited which leads to a lot of research involved in performing basic operations at first but this is somewhat reduced when one gets the hang of the tool kit.
Through the project, we learnt a lot about the different encryption schemes as well as the concepts revolving Homomorphic Encryption as well as different concepts such as boot strapping as well as working with C++ felt like breath of fresh air.

4 Contributions

As there were some differences in contributions, as can be seen on the git repository as well, these are listed here.

4.1 Hindrik

1. **Implemented: BGV Numeric multiplication**
2. **Implemented: BGV Cross product**
3. **Implemented: CKKS Addition**
4. **Implemented: BGV Binary comparison**
5. **Implemented: BGV Binary comparison with bootstrapping**
6. **Implemented: CKKS 2x2 Matrix Determinant**
7. **Implemented: CKKS 3x3 Matrix Determinant**
8. **Implemented: CKKS Complex 3x3 Matrix product**
9. **Implemented: UI for almost all examples**
10. **Attempted: GCD algorithm**

4.2 Rishabh

1. **Report: introduction section**
2. **Implemented:Complex number multiplication in CKKS**
3. **Implemented: Complex number rotation in CKKS along with UI**
4. **Contributed : Complex Matrix multiplication with Hindrik**
5. **Tried obtaining magnitude of Complex number, however could not succeed due to lack of square root functionality**

4.3 Andris

1. **Implemented: BGV Quadratic Polynomial calculation**
2. **Implemented: UI for CKKS Quadratic Polynomial calculation**
3. **Implemented: BGV Complex Conjugate**
4. **Implemented: UI for BGV Complex Conjugate**
5. **Documentation: Instruction for running the project**

References

[1] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011. https://eprint.iacr.org/2011/277.

[2] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Report 2016/421, 2016. https://eprint.iacr.org/2016/421.

[3] A. El-Yahyaoui and D. Kettani. *A New Encryption Scheme to Perform Smart Computations on Encrypted Cloud Big Data*, pages 313–320. 01 2019.

[4] S. Halevi and V. Shoup. Algorithms in helib. Cryptology ePrint Archive, Report 2014/106, 2014. https://eprint.iacr.org/2014/106.