National College of Ireland

Bootstrapping-Based Arithmetic Operations Using Homomorphic Encryption

MSc Research Project
MSc in Cyber Security

Ankit Ajay Kumar
Student ID: x21173664

School of Computing
National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ankit Ajay Kumar		
Student ID:	x21173664		
Programme:	MSc Cyber Security	**Year:**	2022-2023
Module:	MSc Research Project		
Supervisor:	Michael Pantridge		
Submission Due Date:	18/09/2023		
Project Title:	Bootstrapping-Based Arithmetic Operations Using Homomorphic Encryption		
Word Count:	5037 words	**Page Count:**	18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ankit Ajay Kumar

Date: 18/09/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Bootstrapping-Based Arithmetic Operations Using Homomorphic Encryption

Ankit Ajay Kumar
x21173664

Abstract

Homomorphic encryption has recently gained prominence in cryptography as a method to perform mathematical operations on encrypted data, only accessible to authorized users. The recent surge of interest in homomorphic encryption has prompted this research endeavour. The notion of manipulating encrypted numerical data while preserving security barriers has fascinated cryptographers. This concept has intrigued researchers, leading to categorizations of encryption schemes as Partial Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SHE), and Fully Homomorphic Encryption (FHE) based on their computational capabilities. This research study focuses on enhancing Gangula S.'s (2022) very basic implementation of the BFV scheme by progressing it to a Fully Homomorphic Encryption scheme by integrating the bootstrapping process using the Microsoft's SEAL library. Bootstrapping is crucial for refining ciphertext noise levels, which ensures successful decryption of the ciphertext which in turn would facilitate further computations on the ciphertexts. Evaluating its success involves comparing noise levels in ciphertexts before and after bootstrapping.

Keywords: Cryptography, Fully Homomorphic Encryption, BFV, Bootstrapping, Microsoft SEAL.

Table of Contents

Table of Figures

1. Introduction

The advent of computers and the internet has introduced a range of challenges, including the security of data both at rest and in transit. Consequently, ensuring secure communication and data protection has become increasingly crucial. Cryptography serves as the solution to these challenges, playing a vital role in safeguarding data transmission across networks and preventing unauthorized access. Its significance in computer science has grown substantially due to the expanding reliance on digital systems for data storage and communication. This tool empowers individuals and businesses to shield their digital assets from online threats, preventing interception and manipulation of private information. The absence of cryptography would expose sensitive data to breaches, endangering the security and privacy of individuals and organizations. Throughout history, cryptography has been employed to secure sensitive information, spanning from military strategies to diplomatic correspondence. In contemporary times, cryptography is a cornerstone of digital security, ensuring secure online transactions and protecting sensitive data. An exemplar of this encryption method is homomorphic encryption, enabling operations on encrypted data without prior decryption.

Cryptographic algorithms in general are divided into two – Symmetric Key Cryptography and Asymmetric Key Cryptography – based on the number of keys used in them for encryption

and decryption process. Figure 1 and figure 2 below shows how basic communication takes place using the symmetric key and asymmetric key cryptography respectively.

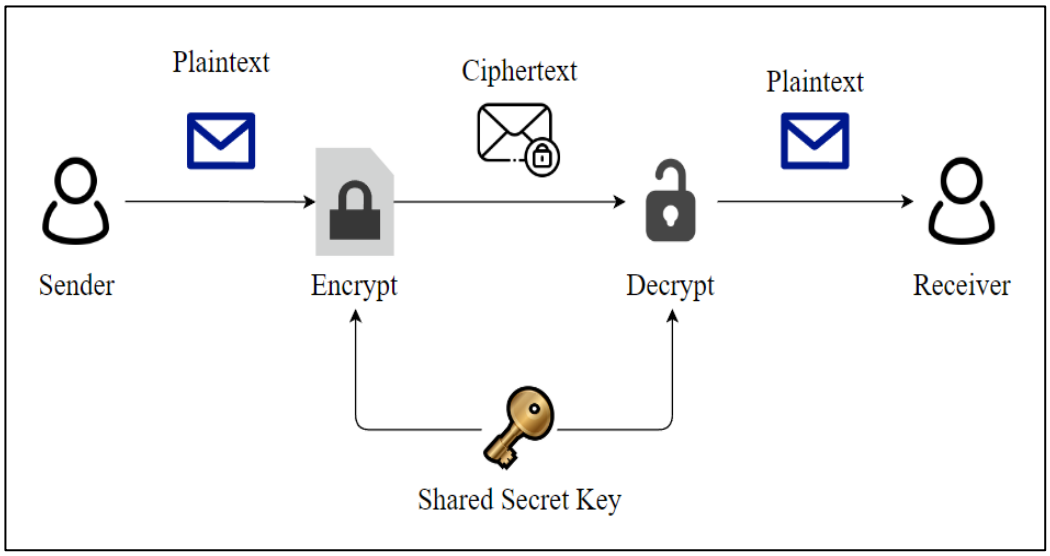

Figure 1: Symmetric Key Cryptography

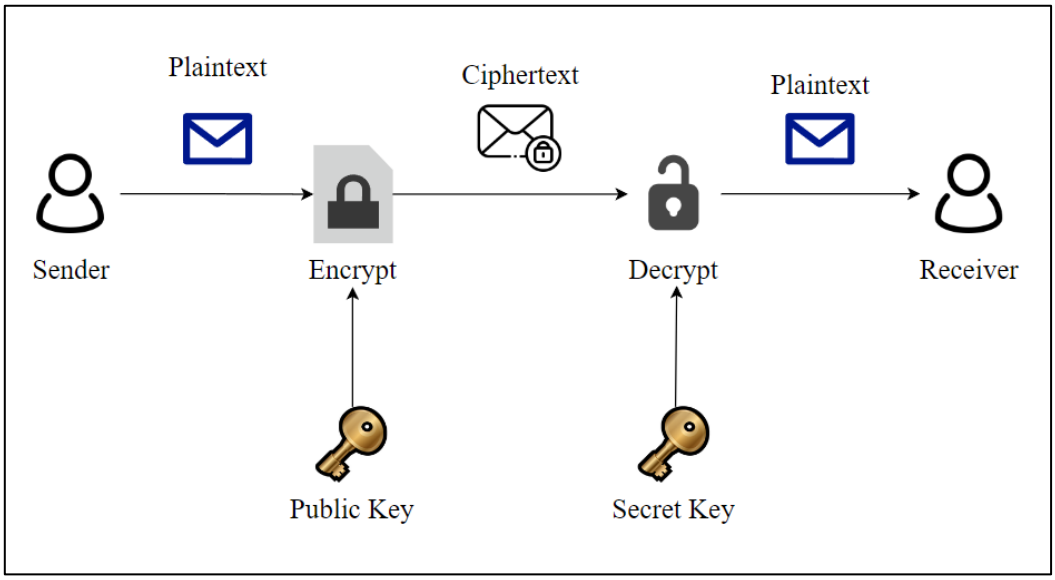

Figure 2: Asymmetric Key Cryptography

The term *'homomorphism'* originates from ancient Greek, where *'homos'* signifies sameness or similarity, and *'morphe'* refers to form or shape as mentioned by Arita, S., Nakasato, S. (2017). Britannica, T. Editors of Encyclopaedia (2008) says that mathematically, it signifies a unique relationship between elements of two algebraic systems like groups, rings, or fields. In cryptography, homomorphism is harnessed for encryption, allowing mathematical operations on encrypted input without preliminary decryption. This concept is relatively novel, granting the ability to conduct operations on encrypted data without compromising its security during processing, and without necessitating prior decryption. Based on the number of consecutive computations feasible before result decryption becomes infeasible, Homomorphic Encryption (HE) divides into three categories: Partial Homomorphic Encryption (PHE) permits a single arithmetic operation (addition or multiplication), Somewhat Homomorphic Encryption (SHE) facilitates multiple arithmetic operations on encrypted data with limitations on successive

computations, and Fully Homomorphic Encryption (FHE) enables limitless computations on ciphertext using a technique called bootstrapping.

All present Homomorphic Encryption (HE) schemes are presently based in the concept of *'noise'*. In essence, Geelen, R. and Vercauteren, F. (2023) say that each ciphertext contains an *'error'* or *'noise'* component that grows as computations progress. Consequently, a mechanism is essential to attenuate this noise and bring it back to a lower level when it surpasses the upper threshold, ensuring continued computations. Bootstrapping encompasses the procedure of refreshing ciphertexts once the noise surpasses the decryption-impossible threshold.

This research project aims to advance the work conducted by Gangula S. (2022). Gangula implemented a rudimentary HE system based on the BFV (Brakerski-Fan-Vercauteren) scheme, making initial strides toward a rudimentary bootstrapping process as recommended in the thesis's future work section. This endeavor will use the Simple Encrypted Arithmetic Library (SEAL), a C++ library developed by Microsoft in Laine, K. (n.d.).

A. Research Question.

The primary aim of this research thesis is to be able to perform computations without worrying about the increase in noise in the ciphertexts after every computation is performed using the basic implementation of the BFV scheme developed by Gangula S. (2022). To achieve this, we need to answer the following research questions.

Q.1. How can we reduce the noise introduced by the HE schemes in order to perform computation on the encrypted data without any restrictions?

B. Outline

Top-down approach is followed to write this research project thesis: Section 2 starts by explaining the division of HE schemes based on the type of bootstrapping process involved and ends by stating the research niche and expected contribution of the research. Section 3 starts by revisiting the existing HE scheme implemented upon which how to implement the bootstrapping process is covered under proposed methodology and system design. Section 4 shows the implementation of the design and proposed methodology. Section 5 discusses the results of the implemented system, and the report ends with Section 6 providing conclusions by the author and future work scope/suggestions in the implemented system.

2. Literature Survey

Depending on the kind of bootstrapping process used, homomorphic encryption schemes are divided into three generations.

A. First Generation HE Schemes

Among the first generation schemes is Gentry's initial concept from 2009 suggested by Gentry, C. (2009). He proposed a new homomorphic encryption scheme based on ideal lattices, which included a bootstrapping process that enabled the scheme to evaluate itself without losing security. The bootstrapping process involved taking an encrypted ciphertext and

transforming it into a new ciphertext that can be evaluated without leaking information about the original plaintext.

The bootstrapping process involved several steps, including encoding the original ciphertext as a polynomial, performing a computation on the polynomial, and then re-encrypting the result using a new key. The process also included a technique called noise reduction, which reduces the amount of noise introduced during the computation. The bootstrapping process is repeated iteratively until the noise is below a certain threshold, ensuring that the scheme remains secure.

The paper presents a detailed description of the bootstrapping process, along with a security analysis and performance evaluation. The scheme is shown to be secure under the Learning With Errors (LWE) assumption, and the bootstrapping process is shown to be efficient enough to make the scheme practical for real-world applications. Due to its lack of advantages over more modern schemes, this scheme is no longer in use in today's world.

B. Second Generation HE Schemes

The bootstrapping process used by second generation schemes is slow and complicated, but they profit from amortizing the re-encryption process over simultaneous SIMD computations and a high number of remaining multiplicative stages. BGV (Brakerski-Gentry-Vaikuntanathan), BFV (Brakerski-Fan-Vercauteren) and CKKS (Cheon-Kim-Kim-Song) are examples of such schemes.

Zvika B., Craig G., and Vinod V.. (2012) proposed a new type of encryption scheme called Leveled Fully Homomorphic Encryption (LFHE), which can evaluate circuits of a fixed depth without the need for bootstrapping. The LFHE scheme is based on the learning with errors (LWE) problem and achieves security against chosen-plaintext attacks. The paper presents the design and analysis of the LFHE scheme, including the security guarantees and efficiency properties. The authors also provide proof of concept implementation of LFHE and demonstrate its efficiency by evaluating simple circuits on encrypted data. The results show that LFHE can evaluate circuits more efficiently than existing bootstrapping based FHE schemes. One of the key features of the LFHE scheme is that it operates in levels, where each level corresponds to a specific circuit depth. This allows the scheme to evaluate circuits of a fixed depth without the need for bootstrapping. The LFHE scheme also includes a key-switching technique that allows for ciphertexts to be transformed from one key to another, which eliminates the need for bootstrapping.

One of the main issues with BGV, which led to the development of BFV, was its inefficiency when dealing with plaintext spaces that were not powers of two. BGV required the plaintext space to be a power of two in order to achieve optimal efficiency, which limited its practicality.

Fan and Vercauteren (2012) presents a new homomorphic encryption scheme (BFV) that is more practical than previous fully homomorphic encryption (FHE) schemes. The scheme is based on the learning with errors (LWE) problem and uses modulus switching, a technique that enables efficient evaluation of circuits on encrypted data but introduces additional noise. The bootstrapping process is then used to reduce the noise level and maintain security. The authors

also propose an optimization technique called modulus switching, that enables faster decryption of the encrypted data, which addressed the limitation of BGV. The technique involves switching the decryption key to a smaller modulus to reduce the computational cost of the decryption operation. The paper also includes an implementation of the scheme and demonstrates its practicality by evaluating several benchmark circuits.

(Cheon et al., 2016) present a new homomorphic encryption scheme that allows computations on real numbers with high precision and efficiency. They argue that existing homomorphic encryption schemes are limited when it comes to performing computations on real numbers due to the noise level that increases with each computation and eventually degrades the security of the system. To address this limitation, the authors propose the approximate homomorphic encryption (AHE) scheme that uses a different approach to encoding and encrypting real numbers. The AHE scheme uses a technique called approximate encoding to represent real numbers as polynomials of bounded degree. It also introduces a new operation called approximate evaluation that enables the evaluation of polynomial functions on encrypted data with high precision and low noise level. However, due to the noise level that increases with each computation, the AHE scheme requires a bootstrapping mechanism to refresh the security parameters and reduce the noise level during computation. The bootstrapping mechanism in the AHE scheme involves transforming ciphertexts to a new modulus with a lower noise level using a process called modulus switching. It is efficient and can be performed in logarithmic time, which makes it suitable for a wide range of applications.

C. Third Generation HE Schemes

Gentry created a very basic homomorphic encryption system based on matrix multiplication, which is where the branch of third generation HE scheme starts. Later, it was discovered that the design allows a quick and easy bootstrapping procedure, which prompted the development of FHEW (Fully Homomorphic Encryption over the Integers with Equality Test) and TFHE (Fully Homomorphic Encryption over the Torus).

(Gentry et al., 2013) introduced a new fully homomorphic encryption scheme called FHEW, which is based on the learning with errors (LWE) problem. The bootstrapping mechanism in FHEW involves refreshing the ciphertexts using a technique called modulus switching. Modulus switching involves transforming the ciphertexts to a new modulus with a lower noise level by computing a linear combination of the ciphertexts with a special random matrix. This process reduces the noise level and allows for additional computations to be performed on the encrypted data.

Brakerski and Vaikuntanathan (2011) presented a new fully homomorphic encryption (FHE) scheme based on the Ring-LWE problem. The authors first introduce the notion of key-dependent messages (KDM), where the encrypted message depends on the secret key used to encrypt it. They show that existing FHE schemes are vulnerable to attacks that exploit this property and propose a new FHE scheme that is secure against such attacks. The authors also present a method for bootstrapping, which allows the scheme to evaluate its own decryption circuit homomorphically. This technique is crucial for achieving fully homomorphic encryption, as it enables the scheme to perform arbitrary computations on encrypted data.

D. Reviewed Papers' Summary

Author	Name of the Paper	Comments
Gentry, C. (2009)	Fully homomorphic encryption using ideal lattices	Gentry's paper introduces the concept of fully homomorphic encryption (FHE) using ideal lattices. While not directly focusing on bootstrapping, it sets the foundation for subsequent papers to explore bootstrapping methods as a means to manage noise accumulation and enhance FHE schemes.
Zvika B., Craig G., and Vinod V.. (2012)	(Leveled) fully homomorphic encryption without bootstrapping.	This paper tackles FHE without relying on bootstrapping, providing alternative strategies to mitigate noise growth and enable computation on encrypted data. It offers insights into avoiding the complexities of bootstrapping and presents techniques to achieve practical FHE without this process.
Fan, J. and Vercauteren, F. (2012)	Somewhat Practical Fully Homomorphic Encryption	While not explicitly centred on bootstrapping, this paper contributes to the discourse on noise management and efficiency in FHE schemes. It may contain valuable insights into methods that reduce noise accumulation, indirectly influencing bootstrapping approaches by enabling smoother, noise-resilient operations.

Cheon, J.H., Kim, A., Kim, M. and Song, Y. (2016)	Homomorphic Encryption for Arithmetic of Approximate Numbers	Although bootstrapping is not the primary focus, this paper's emphasis on handling approximate data could offer innovative ways to address noise-related challenges. These methods might indirectly impact bootstrapping techniques by introducing novel noise reduction or management strategies.
Gentry, C., Sahai, A., Waters, B. (2013)	Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based	This paper introduces a conceptually simpler and faster FHE scheme based on learning with errors. While bootstrapping is not central, the novel approach to FHE might provide insights into more efficient noise management strategies, influencing future bootstrapping implementations to achieve better noise control.
Brakerski, Z., Vaikuntanathan, V. (2011)	Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages	This paper contributes to the development of FHE from the Ring-LWE problem. While not directly discussing bootstrapping, its methods for achieving FHE and addressing security concerns could offer insights into potential bootstrapping techniques or noise reduction strategies, indirectly impacting noise management in FHE schemes.

3. Research Methodology

A. Study of Initial Work

The primary aim of this thesis is to implement the bootstrapping-based arithmetic operations using the HE scheme implemented by Gangula S. (2022) from scratch. For this purpose, the author went through his research and tried to understand his work. It is important to note that the BFV principles and based on RLWE assumptions served as the primary source of inspiration for the HE scheme implemented. Let us understand the scheme in detail below:

The scheme can be divided into 5 steps namely –

- Key Generation
  Two keys, one public key for encryption of the data and one private key for the decryption of the results, are generated before starting the main procedure of computations.
- Encryption
  The formula used for encryption considering the RLWE assumptions was given as follows:

$$R_t = Z_t / (x^n + 1)$$

  Where:
  t = plain-text modulus
  R = RLWE factor
  Z = set of integers
  n = power of 2
  x = positive polynomial
- Evaluation
  The inputs entered by the sender are ready to be computed with arithmetic operations like addition and multiplication. This is all done on the inputs encrypted using the public key of the sender.
- Decryption
  This process takes place at the receiver's end with the help of the private key generated and shared by the sender. Decryption is done in order to verify the results of the computation performed.
- Scheme's Parameters [1]
  # polynomial modulus degree: n = 2**4
  # ciphertext modulus: q = 2**15
  # plaintext modulus: t = 2**8
  # polynomial modulus: poly_mod = np.array([1] + [0] * (n - 1) + [1])

Figure 3 below shows the workflow of the above implemented scheme.

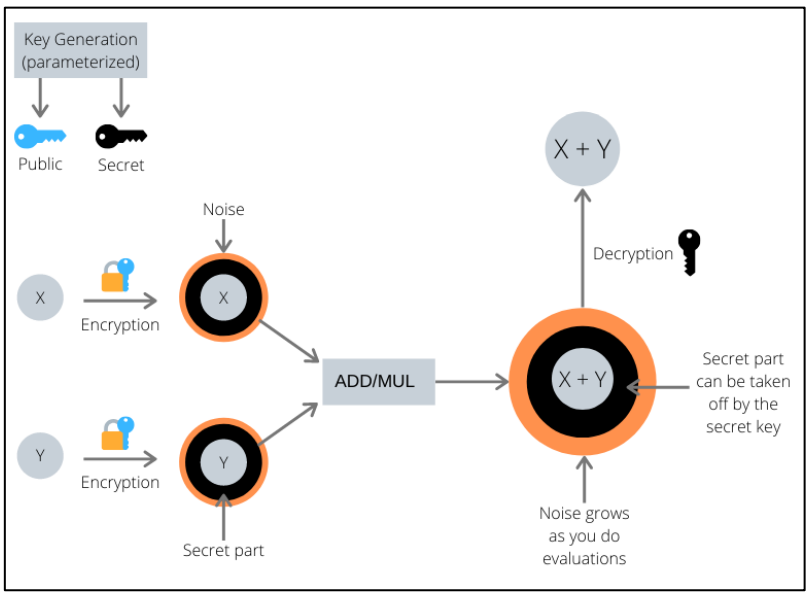

Figure 3: System Workflow of Gangula S. (2022) Scheme

B. Proposed Methodology and System Design

The above HE scheme implemented is only limited to a certain number of continuous computations before the noise in the ciphertext grows so much that it becomes impossible to decrypt the ciphertext to get back the plaintext of the results Gangula S. (2022).

Thus, the author, in this thesis, proposes to implement the bootstrapping process in the scheme in order to make the scheme a Fully Homomorphic Encryption scheme as then the scheme would be able to perform unlimited number of computations on the ciphertext. This would be implemented by using the Simple Encrypted Arithmetic Library (SEAL) developed by MICROSOFT (2023). Below figure 4 shows the high-level flow-chart or design of the proposed system / HE scheme with bootstrapping process.

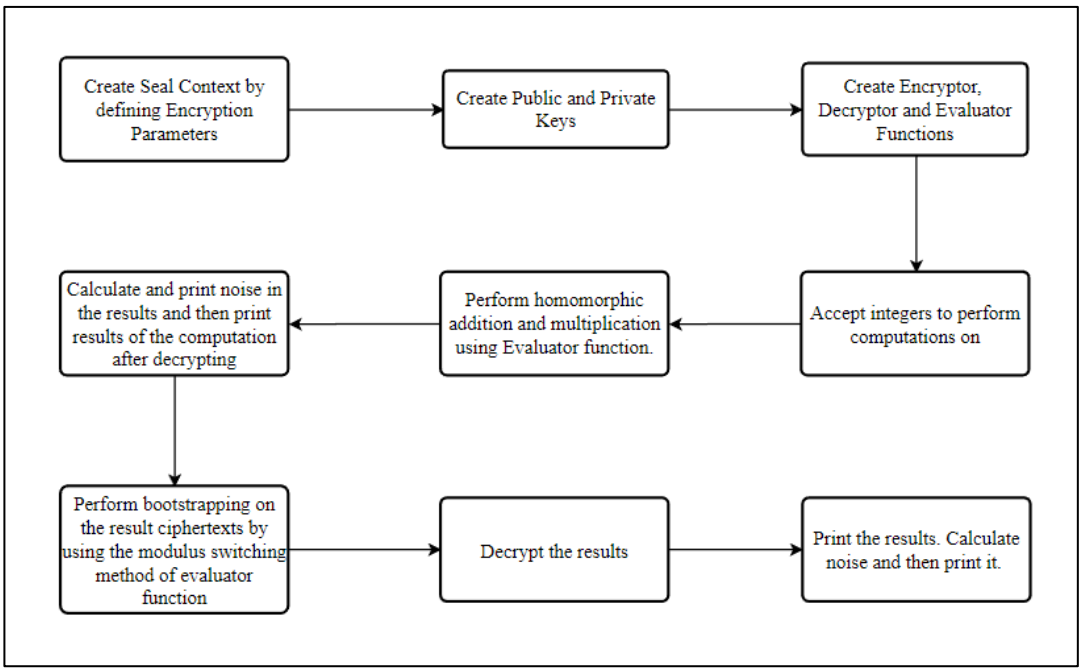

Figure 4: Proposed System Flowchart

4. Implementation

We have implemented the above proposed system using the following tools and technology:

- Programming Language: C++
- Microsoft SEAL library for C++
- IDE: Microsoft Visual Studio 2022

The implementation of the proposed system can be broken down into below following steps:

A. Create SEAL context

To create a SEAL context, we need to define all the encryption parameters for the scheme. We need to define the scheme_type, polynominal_modulus_degree, coefficient_modulus, plaintext_modulus. Then we pass these parameters to the context function.

```
// Create a SEAL context
EncryptionParameters parms(scheme_type::bfv);
parms.set_poly_modulus_degree(8192);
parms.set_coeff_modulus(CoeffModulus::BFVDefault(8192));
parms.set_plain_modulus(PlainModulus::Batching(4096, 20));
SEALContext context(parms);
```

Figure 5: Code Snippet of SEAL Context

In the above figure 5, we can see that *scheme_type* is set to *'BFV', poly_modulus_degree* is set to 8192 which is 2^{13}, *coeff_modulus* is set to again 8192 and the *plain_modulus* is set to 4096 and 20.

The polynomial modulus degree is a parameter that affects the size of the ciphertext and the computational performance of the scheme. In our case, 8192 is chosen. The larger the degree, the larger the size of the ciphertext and the slower the computations, but it also increases the capacity for computations on the ciphertexts and the precision of the computations. So, it's a trade-off between computational efficiency and the capacity/precision of the computations. The coefficient modulus is a parameter that affects the security level of the scheme. The plaintext modulus is a parameter that affects the precision of the computations. The Batching function is a helper function in the SEAL library that generates a prime number for the plaintext modulus as mentioned in MICROSOFT (2023). The arguments to this function are the polynomial modulus degree and the bit size of the prime number. In our case, 4096 is the polynomial modulus degree, and 20 is the bit size of the prime number. So, this function is generating a 20-bit prime number that is suitable for a polynomial modulus degree of 4096.

B. Generate required keys

We will be using asymmetric key cryptography in the proposed scheme. Hence, we will need to generate two keys – Public key and Private key – for the purpose of encryption and decryption. Below figure 6 shows how keys are generated using the SEAL library.

```
// Generate the public and secret keys
KeyGenerator keygen(context);
PublicKey pk;
keygen.create_public_key(pk);
SecretKey secret_key = keygen.secret_key();
```

Figure 6: Code Snippet of Key Generation

C. Create encryptor, decryptor and evaluator functions

In order to perform the encryption and decryption on the plaintexts and ciphertexts respectively, we need to create the encryptor and decryptor functions. The encryptor function accepts context and public key as arguments whereas on the other hand the decryptor function accepts context and the private key as arguments. The evaluator function is initialized which contains methods to perform the homomorphic computations on the ciphertexts. Below figure 7 shows the code snippet for it.

```
// Create an encryptor and decryptor
Encryptor encryptor(context, pk);
Decryptor decryptor(context, secret_key);

// Create an evaluator for homomorphic operations
Evaluator evaluator(context);
```

Figure 7: Code Snippet for encryptor, decryptor and evaluator

D. Perform Homomorphic operations

The evaluator function has methods to perform the homomorphic operations on the encrypted plaintexts. Below figure 8 depicts it.

```
// Perform homomorphic addition
Ciphertext ciphertext_sum;
evaluator.add(ciphertext1, ciphertext2, ciphertext_sum);

// Perform homomorphic multiplication
Ciphertext ciphertext_product;
evaluator.multiply(ciphertext1, ciphertext2, ciphertext_product);
```

Figure 8: Code Snippet of Homomorphic Addition and Multiplication

E. Print results and noise before bootstrapping

We decrypt the results and print the results of the operations performed. We then calculate the noise component of the result ciphertext of both the operations. These are done before implementing the bootstrapping process. Below figure 9 depicts it.

```
// Print the results before bootstrapping
Plaintext plaintext_sum_before;
Plaintext plaintext_product_before;
decryptor.decrypt(ciphertext_sum, plaintext_sum_before);
decryptor.decrypt(ciphertext_product, plaintext_product_before);
cout << "Homomorphic Addition Result (Before Bootstrapping): " << plaintext_sum_before.to_string() << endl;
cout << "Homomorphic Multiplication Result (Before Bootstrapping): " << plaintext_product_before.to_string() << endl;

cout << "Noise in Addition Result Ciphetext (Before Bootstrapping): " << decryptor.invariant_noise_budget(ciphertext_sum) << endl;
cout << "Noise in Multiplication Result Ciphetext (Before Bootstrapping): " << decryptor.invariant_noise_budget(ciphertext_product) << endl;
```

Figure 9: Code Snippet for print results and noise in them

F. Perform bootstrapping process and print results

We perform the bootstrapping process on the resulting ciphertexts of the addition and multiplication operations. We have used the modulus switching method of the evaluator function. In the BFV scheme, modulus switching is used as a method of reducing the noise in the ciphertext. It's a process that involves changing the ciphertext modulus, effectively reducing the range in which the noise can operate. This helps reduce the noise growth and improve the computational efficiency of further operations as said by Peng, Z. (2019). Refer figure 10 below.

```
// Perform bootstrapping
evaluator.mod_switch_to_next_inplace(ciphertext_sum);
evaluator.mod_switch_to_next_inplace(ciphertext_product);

// Decrypt the results after bootstrapping
Plaintext plaintext_sum_after;
Plaintext plaintext_product_after;
decryptor.decrypt(ciphertext_sum, plaintext_sum_after);
decryptor.decrypt(ciphertext_product, plaintext_product_after);

// Print the results after bootstrapping
cout << "Homomorphic Addition Result (After Bootstrapping): " << plaintext_sum_after.to_string() << endl;
cout << "Homomorphic Multiplication Result (After Bootstrapping): " << plaintext_product_after.to_string() << endl;

cout << "Noise in Addition Result Ciphetext (After Bootstrapping): " << decryptor.invariant_noise_budget(ciphertext_sum) << endl;
cout << "Noise in Multiplication Result Ciphetext (After Bootstrapping): " << decryptor.invariant_noise_budget(ciphertext_product) << endl;
```

Figure 10: Code Snippet of bootstrapping process and printing the results

G. Measure code execution time

We have measured the execution time of the entire code to get an idea to the total time taken to perform the entire process proposed above. Refer figure 11 below.

```
using std::chrono::high_resolution_clock;
using std::chrono::duration_cast;
using std::chrono::duration;
using std::chrono::milliseconds;

auto t1 = high_resolution_clock::now();
bfv_scheme_with_seal();
auto t2 = high_resolution_clock::now();

/* Getting number of milliseconds as a double. */
duration<double, std::milli> ms_double = t2 - t1;

std::cout << "Execution time: " << ms_double.count() << "ms\n";
```

Figure 11: Code Snippet for measuring code execution time

5. Results & System Evaluation

We have run the entire C++ code with SEAL implementation in the Visual Studio Code IDE to implement the bootstrapping process in the basic implementation of the BFV scheme for homomorphic computations by Gangula S. (2023).

We start by randomly selected two integers and passed them as plaintexts to the program. The program gets executed and produces results. Refer figure 12 below for the results of the program.

```
1st Integer: 23
2nd Integer: 30
Homomorphic Addition Result (Before Bootstrapping): 53
Homomorphic Multiplication Result (Before Bootstrapping): 690
Noise in Addition Result Ciphetext (Before Bootstrapping): 146
Noise in Multiplication Result Ciphetext (Before Bootstrapping): 114
Homomorphic Addition Result (After Bootstrapping): 53
Homomorphic Multiplication Result (After Bootstrapping): 690
Noise in Addition Result Ciphetext (After Bootstrapping): 102
Noise in Multiplication Result Ciphetext (After Bootstrapping): 95
Execution time: 303.099ms
```

Figure 12: Results/Output of the program

From the above snippet we can see that the bootstrapping process does not affect the operations being performed on the integers. The bootstrapping process involved was performed using the modulus switching method. The resulting ciphertexts have a larger modulus, and modulus switching is used to bring it down to a smaller modulus. This helps reduce the noise growth and improve the computational efficiency of further operations as said by Peng, Z. (2019). The noise in the ciphertext before the bootstrapping process is significantly higher than when after performing the bootstrapping process on the ciphertexts.

6. Conclusion & Future Work

This section marks the end to the proposed research project report and highlights the conclusion made by the author and is followed by the future work suggestions.

A. Conclusion

In this project, we embarked on a journey into the realm of homomorphic encryption. By leveraging the powerful capabilities of the Simple Encrypted Arithmetic Library (SEAL) in the C++ programming language, we successfully implemented homomorphic addition and multiplication operations. These fundamental operations form the cornerstone of secure computation over encrypted data, opening doors to a wide array of privacy-preserving applications. One of the significant challenges we tackled was managing the inherent noise growth that accumulates during homomorphic computations. This challenge was skillfully addressed through the implementation of modulus switching and the bootstrapping process. By employing these techniques, we effectively mitigated the noise growth and prolonged the usability of encrypted data, making it possible to perform multiple operations without compromising the integrity of the results.

Our implementation not only demonstrated the theoretical underpinnings of homomorphic encryption but also showcased its real-world potential. The ability to perform computations on encrypted data without ever revealing the plaintext not only ensures data privacy but also holds promise for secure outsourcing of computations. Our project serves as a steppingstone towards realizing these applications, providing a solid foundation for further research and development in the field. Through this project, we gained invaluable insights into the intricacies of homomorphic encryption, noise management, and the practical considerations of implementing cryptographic protocols. We deepened our understanding of the SEAL library and its capabilities, honing our skills in C++ programming and cryptographic engineering.

In conclusion, our project successfully realized homomorphic addition and multiplication operations using the BFV HE scheme while addressing the challenges posed by noise growth through bootstrapping. This achievement underscores the viability of homomorphic encryption as a means to perform secure computations on sensitive data. As we move forward, the lessons learned and accomplishments achieved during this project will undoubtedly contribute to the advancement of secure and privacy-preserving technologies, paving the way for a more secure digital future.

B. Future Work Suggestions

As the landscape of cryptographic research continues to evolve, several promising avenues emerge for future work to extend and refine the accomplishments of this project. One

compelling direction for future exploration is the optimization of noise management techniques within the context of homomorphic operations. Investigating novel noise reduction strategies, such as error-compensating codes or more sophisticated bootstrapping techniques, could potentially enhance the efficiency and scalability of the encrypted computations, thus expanding the practical utility of the proposed system.

The performance of the implemented system could be fine-tuned through optimization of the codebase and leveraging hardware acceleration, such as utilizing specialized hardware like GPUs or FPGAs. This avenue could significantly boost the processing speed of homomorphic computations, making the system even more amenable to real-time and resource-intensive applications.

From an application perspective, extending the project to encompass real-world use cases, such as secure data analysis in healthcare, finance, or collaborative machine learning, offers exciting prospects. Tailoring the implementation to specific domains and addressing the challenges unique to those domains could yield invaluable insights and practical solutions.

In summation, the completion of this project paves the way for a multitude of future endeavors. The fusion of innovative noise management, incorporation of advanced cryptographic paradigms, performance optimization, domain-specific applications, and rigorous security analysis collectively forms a rich tapestry of opportunities to propel the realms of homomorphic encryption and secure computation into uncharted and transformative territories.

REFERENCES

[1] Gangula S. (2022). COMPUTATION OF NUMBERS USING HOMOMORPHIC ENCRYPTION. Unpublished master's thesis. National College of Ireland.

[2] Arita, S., Nakasato, S. (2017). Fully homomorphic encryption for point numbers. In: Chen, K., Lin, D., Yung, M. (eds.) Inscrypt 2016. LNCS, vol. 10143, pp. 253–270. Springer, Cham

[3] Britannica, T. Editors of Encyclopaedia (2008, March 12). homomorphism. Encyclopedia Britannica. Available at: https://www.britannica.com/science/homomorphism

[4] Geelen, R. and Vercauteren, F. (2023). Bootstrapping for BGV and BFV Revisited. Journal of Cryptology, [online] 36(2). doi: https://doi.org/10.1007/s00145-023-09454-6.

[5] Laine, K. (n.d.). Simple Encrypted Arithmetic Library 2.3.1. [online] WA, USA: Microsoft Research. Available at: https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf.

[6] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178

[7] Fan, J. and Vercauteren, F. (2012). Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive. [online] Available at: https://eprint.iacr.org/2012/144.

[8] Zvika B., Craig G., and Vinod V.. (2012). (Leveled) fully homomorphic encryption without bootstrapping. In Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12). Association for Computing Machinery, New York, NY, USA, 309–325. https://doi.org/10.1145/2090236.2090262

[9] Cheon, J.H., Kim, A., Kim, M. and Song, Y. (2016). Homomorphic Encryption for Arithmetic of Approximate Numbers. Cryptology ePrint Archive. [online] Available at: https://eprint.iacr.org/2016/421

[10] Gentry, C., Sahai, A., Waters, B. (2013). Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In: Canetti, R., Garay, J.A. (eds) Advances in Cryptology – CRYPTO 2013. CRYPTO 2013. Lecture Notes in Computer Science, vol 8042. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40041-4_5

[11] Brakerski, Z., Vaikuntanathan, V. (2011). Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In: Rogaway, P. (eds) Advances in Cryptology – CRYPTO 2011. CRYPTO 2011. Lecture Notes in Computer Science, vol 6841. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-22792-9_29

[12] Wibawa, F., Catak, F.O., Sarp, S. and Kuzlu, M. (2022). BFV-Based Homomorphic Encryption for Privacy-Preserving CNN Models. Cryptography, 6(3), p.34. doi: https://doi.org/10.3390/cryptography6030034.

[13] Carey, A. (2020). On the Explanation and Implementation of Three Open-Source Fully Homomorphic Encryption Libraries. [UNDERGRADUATE HONORS THESES] Available at: https://scholarworks.uark.edu/cgi/viewcontent.cgi?article=1074&context=csceuht.

[14] MICROSOFT (2023). Microsoft SEAL. [online] GitHub. Available at: https://github.com/microsoft/SEAL/tree/main [Accessed 12 Jul. 2023].

[15] Peng, Z. (2019). Danger of using fully homomorphic encryption: A look at Microsoft SEAL. [online] arXiv.org. doi: https://doi.org/10.48550/arXiv.1906.07127.