

# Configuration Manual

MSc Research Project  
MSc Cybersecurity

Suchin John  
Student ID: 23218266

School of Computing  
National College of Ireland

Supervisor:    Liam Mccabe

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Suchin John  
**Student ID:** 23218266  
**Programme:** MSc Cybersecurity **Year:** 2025  
**Module:** MSc Practicum/Internship part 2  
**Lecturer:** Liam McCabe  
**Submission Due Date:** 29/1/2025  
**Project Title:** How Can Homomorphic Encryption Be Used In Healthcare Industries  
**Word Count:** 1060 **Page Count:** 8

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:** Suchin john

**Date:** 29/1/2025

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

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

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

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Suchin John  
Student ID: 23218266

## 1 Introduction

This configuration manual contains all the information regarding how this research project was completed. In the second section, it discusses the experimental setup that was used and the detailed specifications of the hardware. The different software and technology that was used for the implementation is shown in the third section. This manual provides step by step instruction on how to install and set up each and every software. It provides links from where the different software could be downloaded and screen shots that show how to install and set them up.

## 2 Experimental Setup

The research project was carried out on a personal laptop. The detailed specifications of the laptop are mentioned below:

- The laptop model: Lenovo LOQ.
- Operating system: Windows 11 Version- 10.0.22631 Build 22631
- Processor: 12th Gen Intel(R) Core(TM) i5-12450H, 2000 Mhz, 8Core(s), 12 Logical Processor(s).
- Installed Physical Memory (RAM): 16.0 GB
- Total Virtual Memory: 32.8 GB
- System Type: 64-bit operating system, x64-based processor
- GPU: NVIDIA GeForce RTX 3050 6GB
- Storage: 500GB SSD

## 3 Software and Technology used for Implementation

The primary software and technological requirements to carry out this research is:

- Microsoft SEAL- which is used for Fully Homomorphic Encryption
- C++- The programming language most suited to Microsoft SEAL

It can be downloaded at this link: <https://code.visualstudio.com/download>

Visual Studio Code Docs Updates Blog API Extensions FAQ GitHub Copilot

Version 1.95 is now available! Read about the new features and fixes from October.

# Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.

Windows

Windows 10, 11

User Installer [Download](#)

System Installer [Download](#)

.zip [Download](#)

CLI [Download](#)

.deb

Debian, Ubuntu

.rpm

Red Hat, Fedora, SUSE

.deb [Download](#) [ARM64](#) [ARMv8](#)

.rpm [Download](#) [ARM64](#) [ARMv8](#)

.tar.gz [Download](#) [ARM64](#) [ARMv8](#)

Snap [Download](#)

CLI [Download](#) [ARM64](#) [ARMv8](#)

Mac

macOS 10.15+

.zip [Download](#) [Apple silicon](#) [Universal](#)

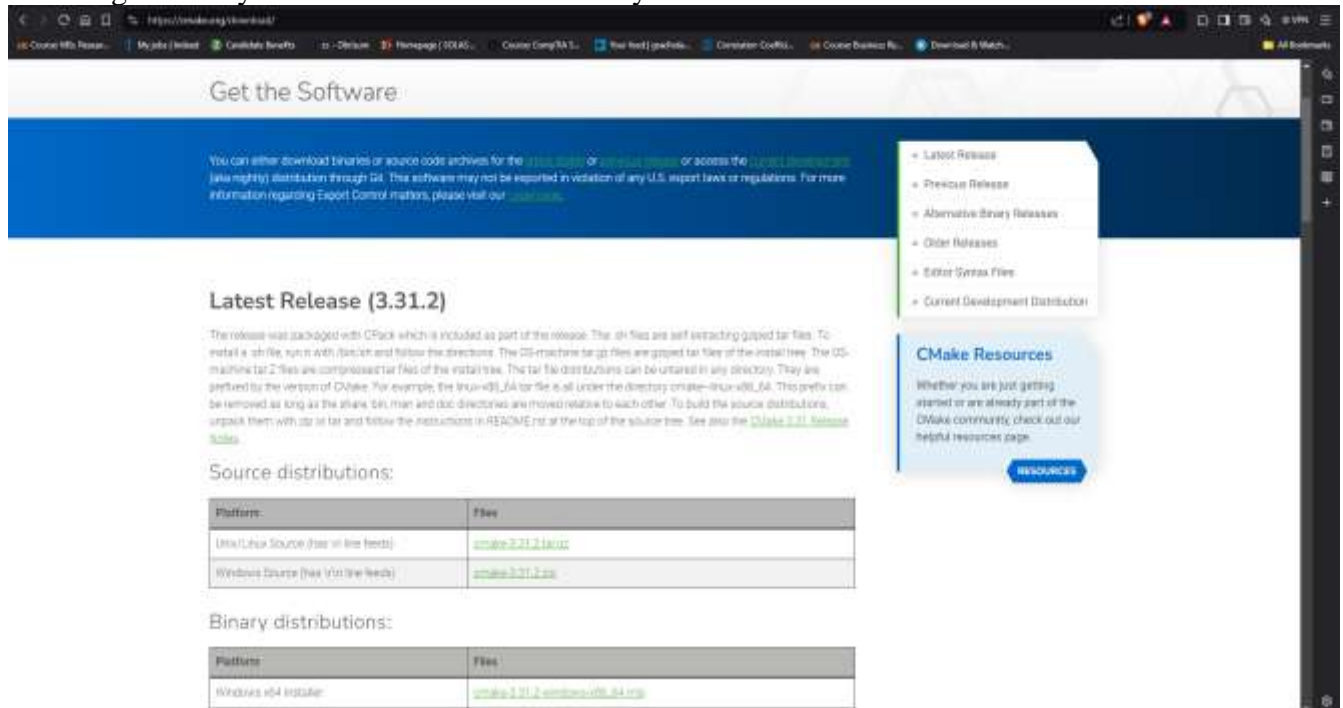
CLI [Download](#) [Apple silicon](#)

Once Visual Studio Code is set up. We will need to install the C++ extension in order to compile and run our code. Microsoft SEAL is designed to be run on C++. We can install the C++ extension by searching for it in the extensions section. The version of C++ that I used is v1.3.0

(Fig. 2)

The next step is to install CMake. CMake is build system generator that simplifies the process of building, configuring, and managing the compilation of complex software projects like Microsoft SEAL. CMake can be downloaded from the listed website:

<https://cmake.org/download/> The version that is used in this project is version 3.31.2 While installing CMake you should add Cmake to the system PATH.



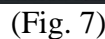
(Fig. 3)

Once Cmake is set up, the next step will be to install Git into our system. Git allows you to download (clone) the latest source code of Microsoft SEAL from its official repository on GitHub. It ensures that you have an up-to-date version of the library. Git can be installed from the website listed here : <https://git-scm.com/downloads/win> While installing and setting up Git, it is also necessary to add Git to the system PATH

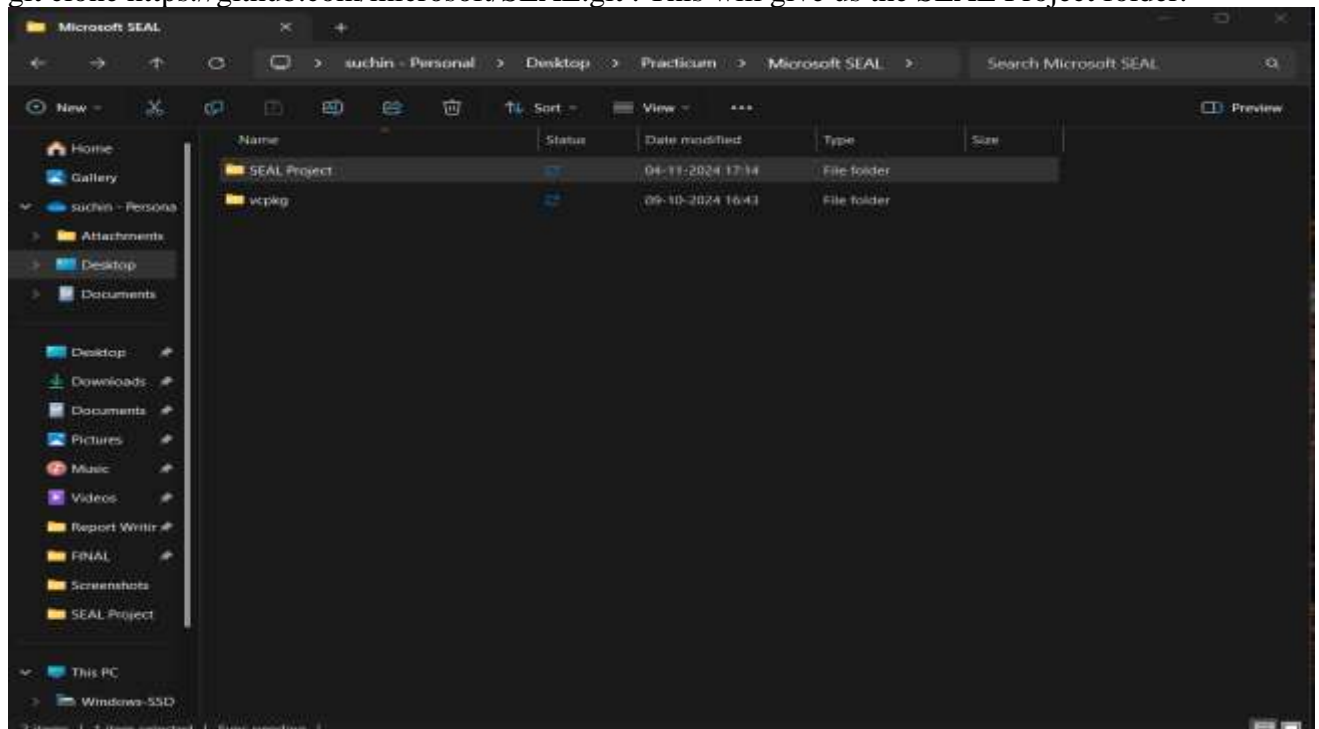


(Fig. 5)

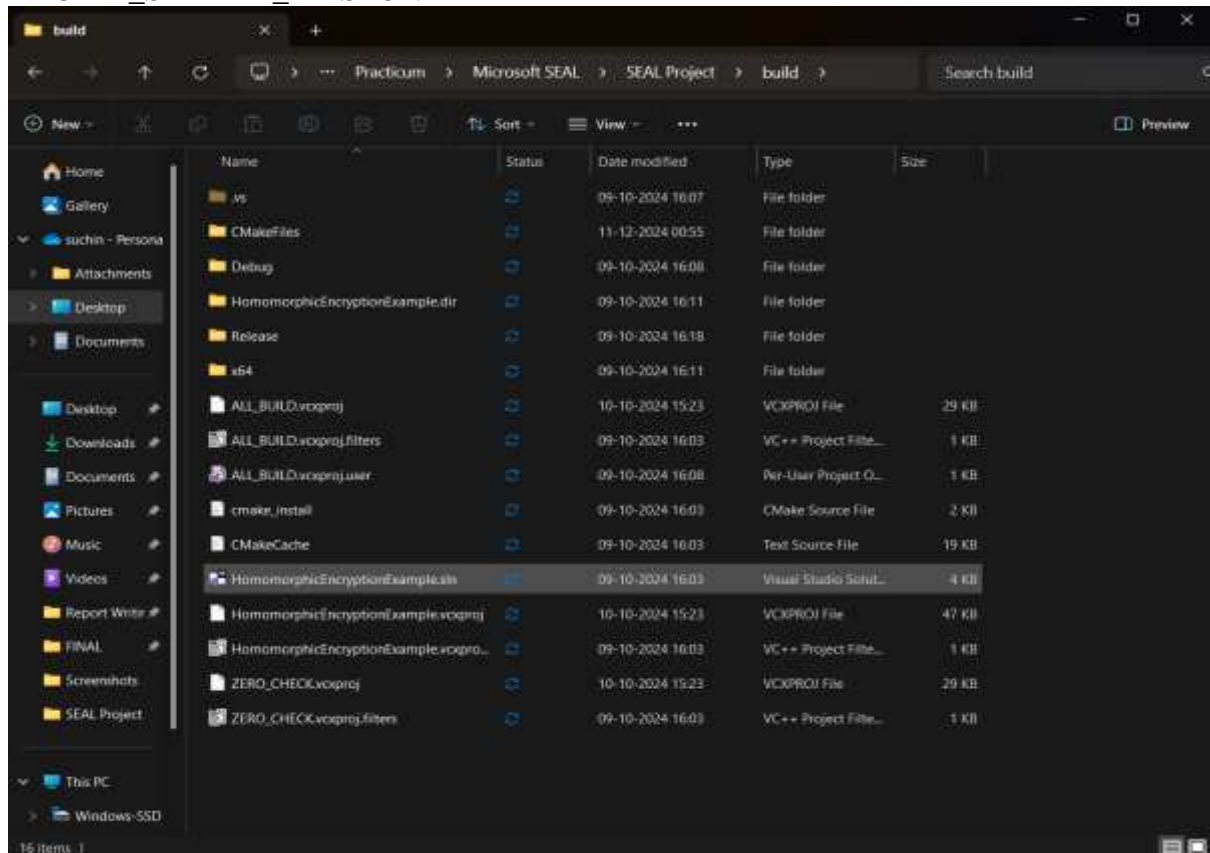
4



(Fig. 8)

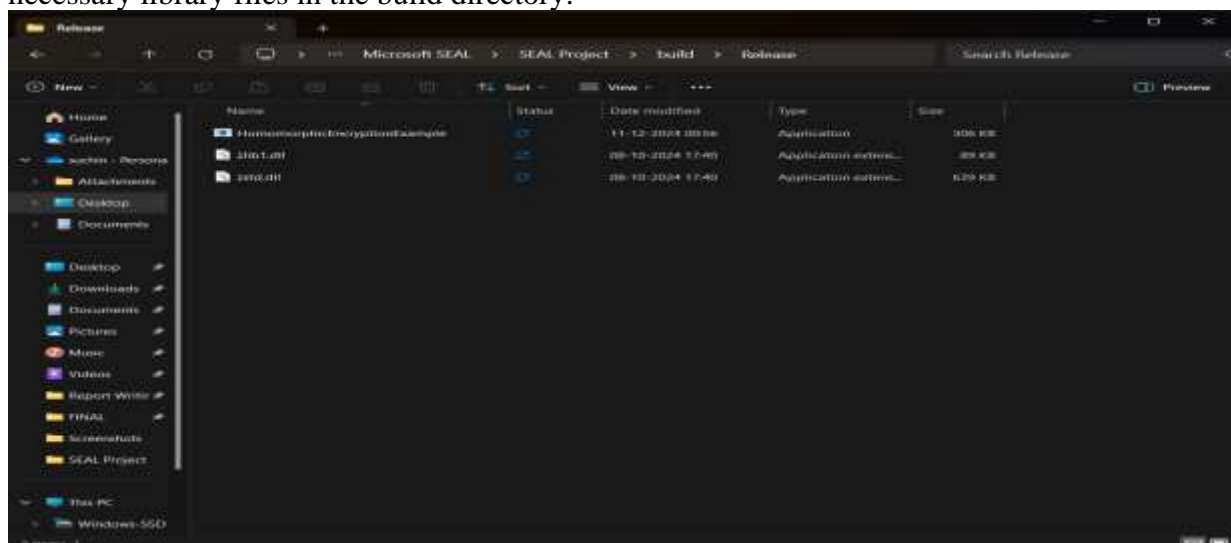


Navigating into the SEAL folder. We have to create a folder called build. Once that is done, the next step is to Run CMake to configure the build. Make sure you're in the build directory, Open the terminal and use the command: `cmake .. -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=ON`



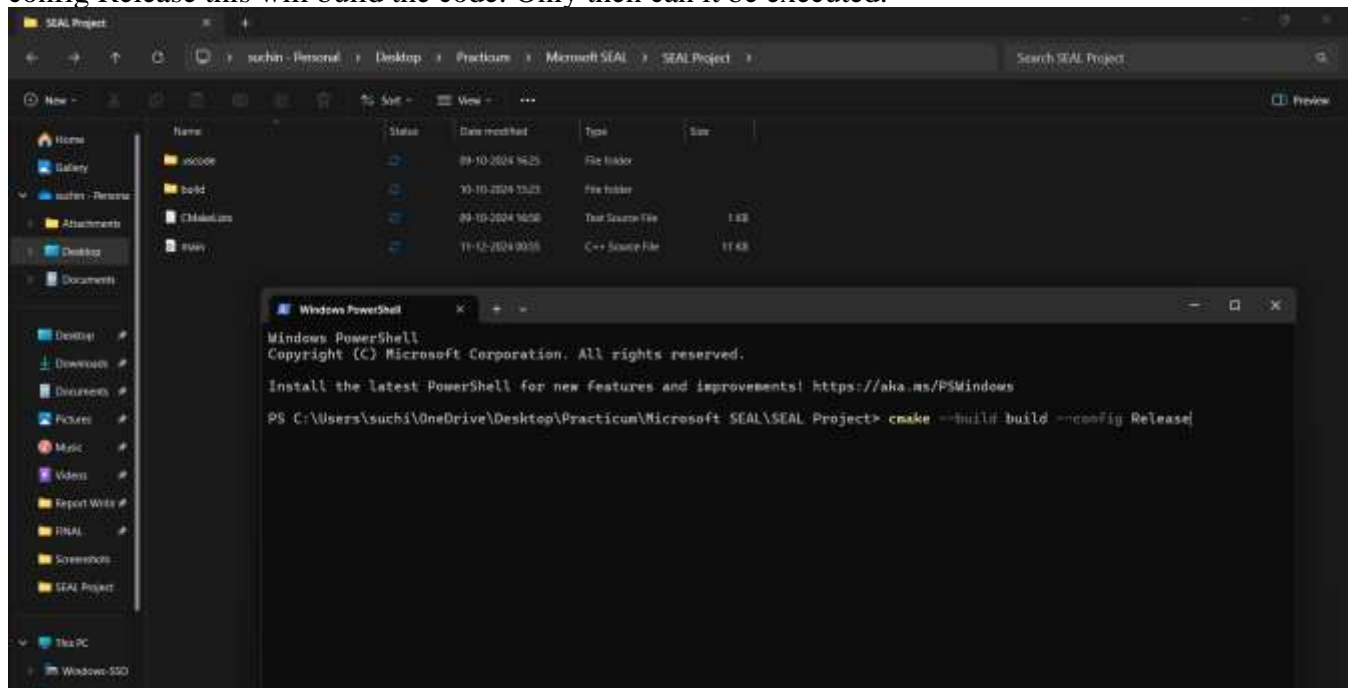
(Fig. 9)

The next step is to build the library using the preferred tool, in this case it is Visual Studio Code. This can be done by using CMake which we installed earlier. Using the command: `cmake --build . --config Release` This command compiles the source files and generates the necessary library files in the build directory.



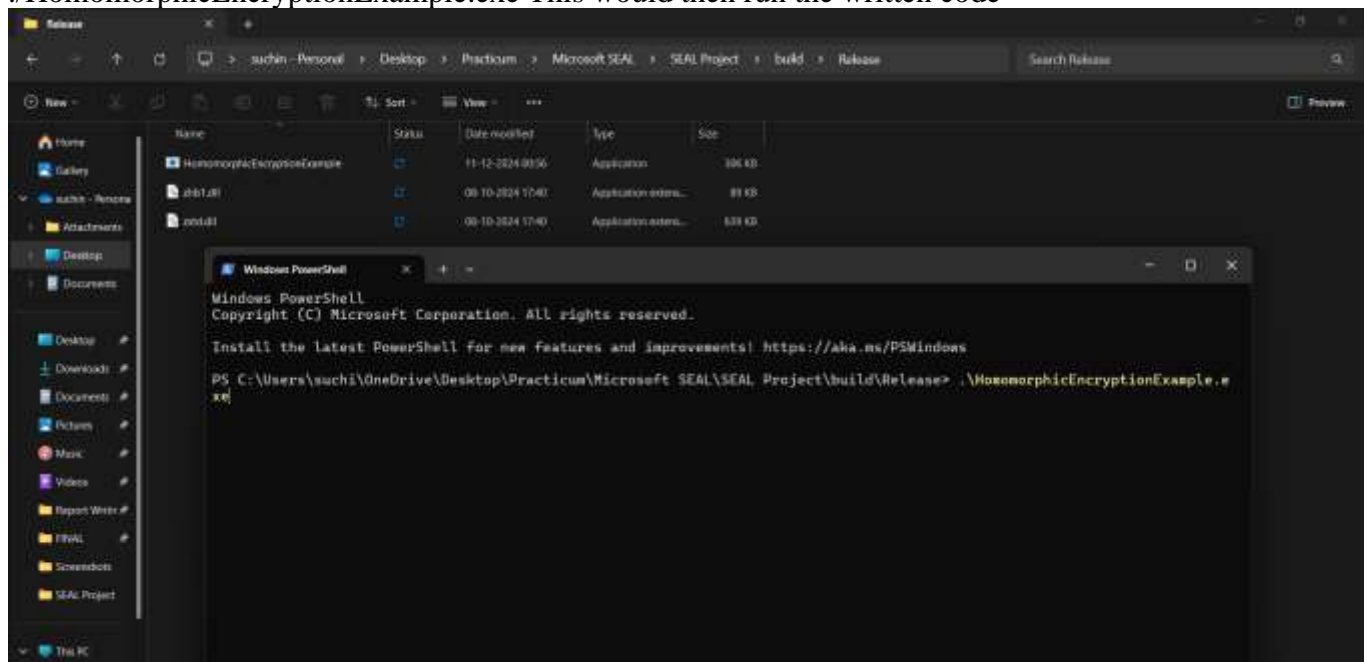
(Fig. 10)

Now that all the prerequisites are complete, u can begin coding. The main C++ code will be located in the SEAL project folder. Once u have written the code, in order to compile and build it you will need to open the terminal and type in the command: `cmake --build build --config Release` this will build the code. Only then can it be executed.



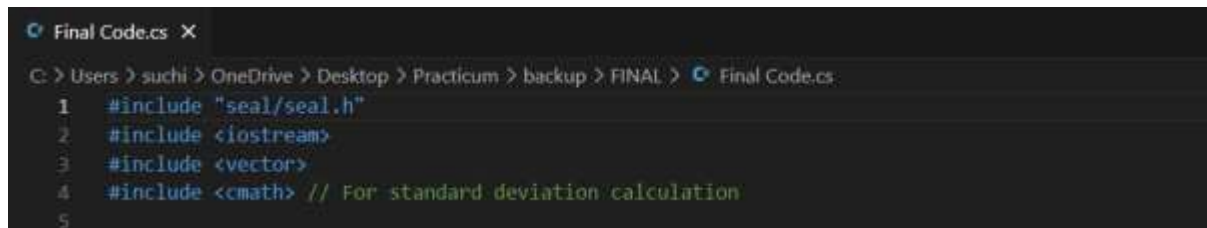
(Fig. 11)

Once the code is compiled and built. We navigate to the build folder and the release folder within it. There we open the terminal and use the command: `./HomomorphicEncryptionExample.exe` This would then run the written code



(Fig. 12)

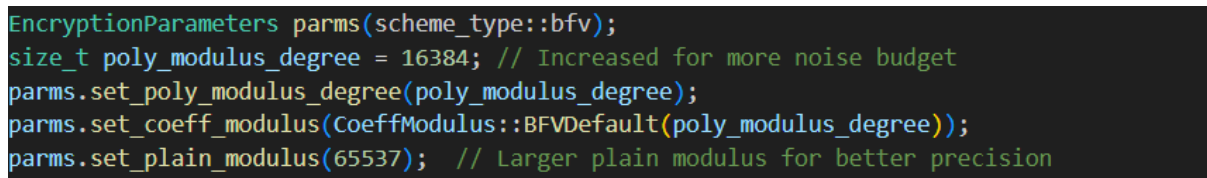
In the next few steps, we shall focus on the necessary components that u must set up when staring to code. The first part is importing the necessary seal libraries. Always import seal/seal.h as it provides essential encryption primitives.



```
Final Code.cs X
C:\Users\suchi> OneDrive\Desktop\Practicum\backup\FINAL> Final Code.cs
1 #include "seal/seal.h"
2 #include <iostream>
3 #include <vector>
4 #include <cmath> // For standard deviation calculation
5
```

(Fig. 13)

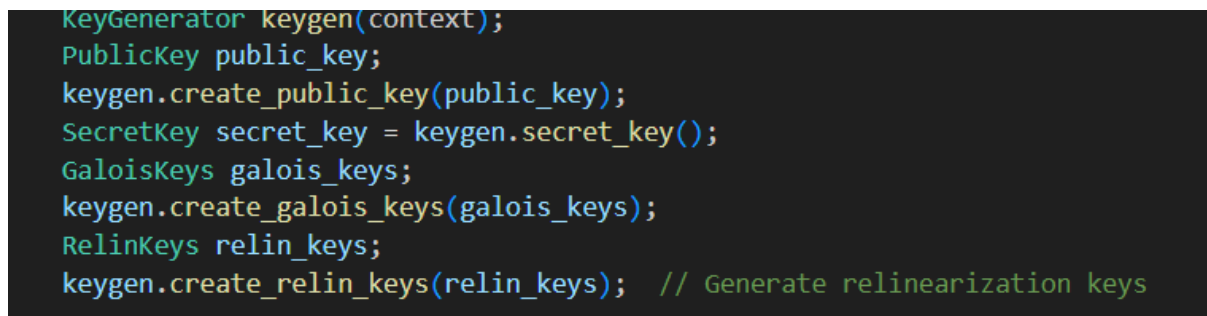
Setting up the parameters is another integral step. Set poly\_modulus\_degree to a power of 2 (e.g., 2048, 4096, 8192, 16384). And also Ensure plain\_modulus is prime and sufficiently large to prevent overflow during computations. The scheme\_type is used for specifying the type of encryption u need. The one used in this project is BFV.



```
EncryptionParameters parms(scheme_type::bfv);
size_t poly_modulus_degree = 16384; // Increased for more noise budget
parms.set_poly_modulus_degree(poly_modulus_degree);
parms.set_coeff_modulus(CoeffModulus::BFVDefault(poly_modulus_degree));
parms.set_plain_modulus(65537); // Larger plain modulus for better precision
```

(Fig. 14)

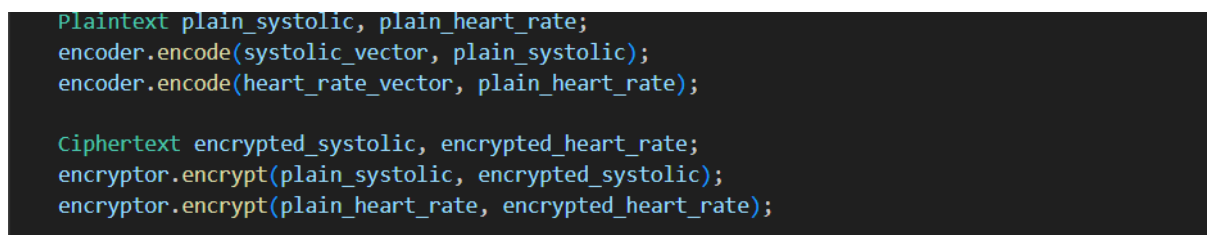
The next step is to generate the keys that will be used for encryption. the different keys that are used in this project are: Public Key For encryption. Secret Key For decryption. Galois Keys For data rotation and Relin Keys For efficient multiplication.



```
KeyGenerator keygen(context);
PublicKey public_key;
keygen.create_public_key(public_key);
SecretKey secret_key = keygen.secret_key();
GaloisKeys galois_keys;
keygen.create_galois_keys(galois_keys);
RelinKeys relin_keys;
keygen.create_relin_keys(relin_keys); // Generate relinearization keys
```

(Fig. 15)

This project encrypts the data by first encoding the plaintext into vectors for homomorphic encryption. This encoded data is then encrypted into ciphertexts. Something to keep note of is to ensure that the data fits into the slot\_count of the encoder. The slot\_count is dependant on the poly\_modulus\_degree



```
Plaintext plain_systolic, plain_heart_rate;
encoder.encode(systolic_vector, plain_systolic);
encoder.encode(heart_rate_vector, plain_heart_rate);

Ciphertext encrypted_systolic, encrypted_heart_rate;
encryptor.encrypt(plain_systolic, encrypted_systolic);
encryptor.encrypt(plain_heart_rate, encrypted_heart_rate);
```

(Fig. 16)

Conducting homomorphic encryption is the heart of this research project. It performs homomorphic encryption ie compute cipher text without needing to decrypt it. Something to keep in mind is to relinearization after squaring or multiplying. This is to manage the cipher text size

```
evaluator.square(encrypted_systolic, encrypted_systolic_square_sum);
evaluator.relinearize_inplace(encrypted_systolic_square_sum, relin_keys);
evaluator.square(encrypted_heart_rate, encrypted_heart_rate_square_sum);
evaluator.relinearize_inplace(encrypted_heart_rate_square_sum, relin_keys);
evaluator.multiply(encrypted_systolic, encrypted_heart_rate, encrypted_cross_product_sum);
evaluator.relinearize_inplace(encrypted_cross_product_sum, relin_keys);
```

(Fig. 17)

The aggregation in this project is done using rotation which is achieved through the use of the previously generated galois\_keys. Rotation in homomorphic encryption is necessary because of the way the data is encoded into slots in the polynomial structure of the ciphertext. This approach allows summing up all elements in an encrypted vector without decrypting it

```
evaluator.rotate_rows(encrypted_systolic, static_cast<int>(i), galois_keys, rotated_systolic);
evaluator.rotate_rows(encrypted_heart_rate, static_cast<int>(i), galois_keys, rotated_heart_rate);
evaluator.rotate_rows(encrypted_systolic_square_sum, static_cast<int>(i), galois_keys, rotated_systolic_square);
evaluator.rotate_rows(encrypted_heart_rate_square_sum, static_cast<int>(i), galois_keys, rotated_heart_rate_square);
evaluator.rotate_rows(encrypted_cross_product_sum, static_cast<int>(i), galois_keys, rotated_cross_product);
```

(Fig. 18)

The final part of code is to decrypt and decode the ciphertext. This Converts ciphertext back to plaintext and decodes it into readable data. Another thing to keep not of while decrypting is to Match encoding and decoding slots during data processing.

```
decryptor.decrypt(encrypted_systolic_sum, plain_systolic_sum);
decryptor.decrypt(encrypted_heart_rate_sum, plain_heart_rate_sum);

vector<uint64_t> decoded_systolic_sum, decoded_heart_rate_sum;
encoder.decode(plain_systolic_sum, decoded_systolic_sum);
encoder.decode(plain_heart_rate_sum, decoded_heart_rate_sum);
```

(Fig. 19)

## 4 References

Visual Studio Code (2016). Visual Studio Code. [online] Visualstudio.com. Available at: <https://code.visualstudio.com/Download>.

Cmake.org. (2018). CMake. [online] Available at: <https://cmake.org/>.

Git-scm.com. (2024). Git - Downloading Package. [online] Available at: <https://git-scm.com/downloads/win>.

GitHub. (2022). Microsoft SEAL. [online] Available at: <https://github.com/Microsoft/SEAL>.