PRIVACY PRESERVING CLOUD COMPUTATION FOR DATA COLLECTED FROM MICROCONTROLLERS

by

Georgios Daoutis

BSc, Informatics Engineering, Hellenic Mediterranean University, 2020

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

2022

Approved by:

Major Professor
Georgios Kornaros

Abstract

The growth of the internet of things has given us the ability to monitor and control various situations from distance, mostly by collecting data from the edge, then storing and distributing them with the use of servers. This data can be the location of a person, medical data, industrial data, or other sensitive information, so we must protect them from unauthorized users. If the server where the data will be stored is semi-trusted, then we must protect take action to protect those data. As semi-honest, we referred to a party that adheres to the protocol correctly while also keeping a log of all its intermediate calculations. The proposed solution to this problem is the use of homomorphic encryption which give us the ability to perform operations on encrypted data such as additions and multiplications without the need of decrypting the data first. This makes homomorphic cryptography ideal for this use because the server can perform operations and at the same time, he cannot have access to the data.

In this project, we will build the above scenario for data collected from microcontrollers. The microcontroller will have various data stored on its local memory and when the user wants to offload them and send them to the cloud for storage, the microcontroller will first encrypt those data with homomorphic encryption and then send them to the cloud. The microcontroller will connect with the outside world via NFC. When the user wants to extract some information from those data, he will ask the server to execute an algorithm on those data and then send back the result encrypted that only the user can decrypt with the android application using his secret key. The query implemented is an encrypted inference based on a dataset that determines the presents of humans in a room. For that reason, we trained a perceptron and designed a that can run on the server without the server can access the private values that the board collected.

Περίληψη

Η ανάπτυξη του Διαδικτύου των πραγμάτων μας έδωσε τη δυνατότητα να παρακολουθούμε και να ελέγχουμε διάφορες καταστάσεις από απόσταση, κυρίως συλλέγοντας δεδομένα από τερματικες συσκευες, αποθηκεύοντας και διανέμοντάς τα με τη χρήση διακομιστών. Αυτά τα δεδομένα μπορεί να είναι η τοποθεσία ενός ατόμου, ιατρικά δεδομένα, βιομηχανικά δεδομένα ή άλλες ευαίσθητες πληροφορίες, επομένως πρέπει να τα προστατεύουμε. Εάν ο διακομιστής στον οποίο θα αποθηκευτούν τα δεδομένα είναι ημι-έμπιστος, τότε πρέπει να λάβουμε μέτρα για την προστασία αυτών των δεδομένων. Ως ημι-έμπιστος, αναφερομαστε σε καποιον που τηρεί σωστά το πρωτόκολλο, ενώ διατηρεί επίσης ένα αρχείο καταγραφής όλων των ενδιάμεσων υπολογισμών του. Η προτεινόμενη λύση σε αυτό το πρόβλημα είναι η χρήση ομομορφικής κρυπτογράφησης που μας δίνει τη δυνατότητα να εκτελούμε πράξεις σε κρυπτογραφημένα δεδομένα όπως προσθέσεις και πολλαπλασιασμούς χωρίς να χρειάζεται πρώτα να αποκρυπτογραφήσουμε τα δεδομένα. Αυτό καθιστά την ομομορφική κρυπτογραφία ιδανική για αυτή τη χρήση, επειδή ο διακομιστής μπορεί να εκτελέσει λειτουργίες και ταυτόχρονα, δεν μπορεί να έχει πρόσβαση στα δεδομένα.

Σε αυτη την διπλωματικη, δημιουργήσαμε το παραπάνω σενάριο για δεδομένα που συλλέγονται από μικροελεγκτές. Ο μικροελεγκτής θα έχει διάφορα δεδομένα αποθηκευμένα στην τοπική του μνήμη και όταν ο χρήστης θέλει να τα στείλει στο cloud για αποθήκευση, ο μικροελεγκτής θα κρυπτογραφήσει πρώτα αυτά τα δεδομένα με ομομορφική κρυπτογράφηση και στη συνέχεια θα τα στείλει στο cloud. Ο μικροελεγκτής συνδέεται με τον έξω κόσμο μέσω NFC.

Όταν ο χρήστης θέλει να εξαγάγει κάποιες πληροφορίες από αυτά τα δεδομένα, θα ζητήσει από τον διακομιστή να εκτελέσει έναν αλγόριθμο σε αυτά τα δεδομένα και στη συνέχεια θα στείλει το αποτέλεσμα κρυπτογραφημένο που μόνο ο χρήστης μπορεί να αποκρυπτογραφήσει με την εφαρμογή android και χρησιμοποιώντας το μυστικό κλειδί του. Επισεις σχεδιάσαμε και υλοποιησαμε εναν τροπο για να επεξεργαστούμε ομοιομορφικα τα δεδομενα. Για αυτόν τον λόγο, εκπαιδεύσαμε ένα perceptron και σχεδιάσαμε ένα αλγοριθμο που μπορεί να τρέχει στον server το perceptron χωρίς να έχει πρόσβαση στις ευαισθητες πληροφοριες που συνέλεξε η πλακέτα.

Table of Contents

List of Figures

List of Abbreviations and Symbols

HE	Homomorphic Encryption
CKKS	Cheon-Kim-Kim-Song
IoT	Internet of Things
MLP	Multi-Layer Perceptron
NFC	Near Field Communication
GPO	General Purpose Output
RSA	Rivest–Shamir–Adleman
LWE	Learning With Errors
PaaS	Platform as a Service
SaaS	Software as a Service
IaaS	Infrastructure as a Service
WSN	Wireless Sensor Networks

1 Introduction

The move to the cloud has been accelerated during the past few years. The processing power and storage made available by the cloud providers are used by millions of users, both businesses and people alike, offering a variety of services that make our lives easier [1]. The IoT in particular has grown significantly. Nowadays, the services that we can benefit from, are ranging from simple databases for storage to complete stand-alone systems for IoT management that can store, process data, and make automated decisions. This kind of applications will make the monitoring and controlling of various events more broadly available with less cost. As a result, we can benefit in many areas like precision farming, smart cities , etc... IoT and many other applications that rely on the cloud will undoubtedly begin to play a bigger role in our personal life soon [2]. Sensitive data that are collected by the IoT devices are sent to the cloud providers and therefore we need to protect them from unauthorized access [3]. Using cryptographic algorithms like symmetric and asymmetric encryption we can achieve limited security because the data must still be decrypted on the server to be processed. This means that we need to have trust in the cloud provider for proper data management.

In recent years as computers have become more powerful and cryptography has evolved, more techniques are being continuously developed that can solve problems that until recently were impossible to be solved, enabling us to have complete control over our data. One of these techniques is homomorphic encryption [4]. Data that has been encrypted using homomorphic encryption can be analyzed or changed without revealing information to anyone. Similar to other types of encryptions, homomorphic encryption encrypts data using a public and a secret key. After the manipulation of the data is completed, only the person who owns the private key can access the encrypted data. Homomorphic encryption will play an important role as it will allow us to have full control over our data but there is still research to be done as it is a new technology. Some issues that prevent us from using this technology in IoT are the large size of the cyphertext data generated and the overhead which burdens the constrained IoT devices [5].

1.1 Contribution

The purposes of this thesis are the design and development of an IoT application while exploring the possibilities of homomorphic encryption in embedded systems and the extraction of information from the data collected while maintaining the privacy of users' data on the cloud. More specifically the infrastructure will consist of an embedded system that will encrypt homomorphically the collected values from sensors using the SEAL-Embedded library[6]. Those values are the temperature of a room, light and sound intensity, PIR, and CO2. Then the ciphertext will be collected through the NFC android app and will be sent to the cloud for storage. For the server to process data and extract information, we trained a perceptron that can determine if there was a human presence at a given time. For the server to be able to calculate the output of the perceptron we also design and implement an algorithm that multiplies an encrypted matrix with one that is not encrypted, the matrices are the wights of the perceptron that are in plaintext form and the other matrix is the input that needs to be multiplied with the weights matrices, this way the values that the board encryped are safe from leaking or to be used from the server for other purposes. Finally, the owner can make queries to the server by using the android app that was developed. The answers to the queries made by the user are sent back to him encrypted.

1.2 Outline

The thesis's outline is presented below, and the important features of each chapter are emphasized.

Chapter 1 - Introduction - An overview is provided in the Introduction section. We mention why homomorphic cryptography is important in IoT and also describe the work developed.

•**Chapter 2** – **Internet of things**: Analysis on the structure of the Internet of Things.

•**Chapter 3 – Cloud:** In this chapter we describe what a Cloud is and the services that it offers as well as concerns about the privacy of the users.

•**Chapter 4 - Homomorphic encryption:** Here we analyze what homomorphic cryptography is and delve into the analysis of the CKK scheme which we used in this work.

•**Chapter 5 - Pattern recognition:** We describe what pattern recognition is and the process we follow to build such a system. We also describe the Multi-Layer Perceptrons algorithm.

•**Chapter 6 – Related work:** Discussion about homomorphic encryption and pattern recognition that relates to our proposed work.

•**Chapter 7 – Implementation:** Detailed performance analysis on the system we implemented.

•**Chapter 8 – Conclusion:** Conclusions and improvements that we can implement in the future.

2 Internet of things

In general, the Internet of Things (IoT) is the interaction between people, objects- devices, and the internet [13]. The IoT is a collection of heterogeneous devices that can connect exchange, and transfer data with other connected devices usually over the internet or inside a local network. These devices are embedded with sensors, software, and other related technologies. Examples of IoT applications include everything from basic sensors for measuring and reporting the temperature to sophisticated industrial equipment for enabling automated control [14]. Healthcare, manufacturing, and agriculture are just a few industries that are quickly adopting IoT as it allows the collection of useful data that can be used for better usage of the available resources or on-time notification of various events. The demand for IoT devices is rising even in regular households. According to Forbes [15] by the year 2025, the IoT industry is expected to have an economic impact of $11 trillion. In a report released in 2016, Statista Research Department predicted that 75 billion IoT devices would be available to consumers worldwide by the year 2025. The central concept of the Internet of Things is defined by a three-layer architecture.

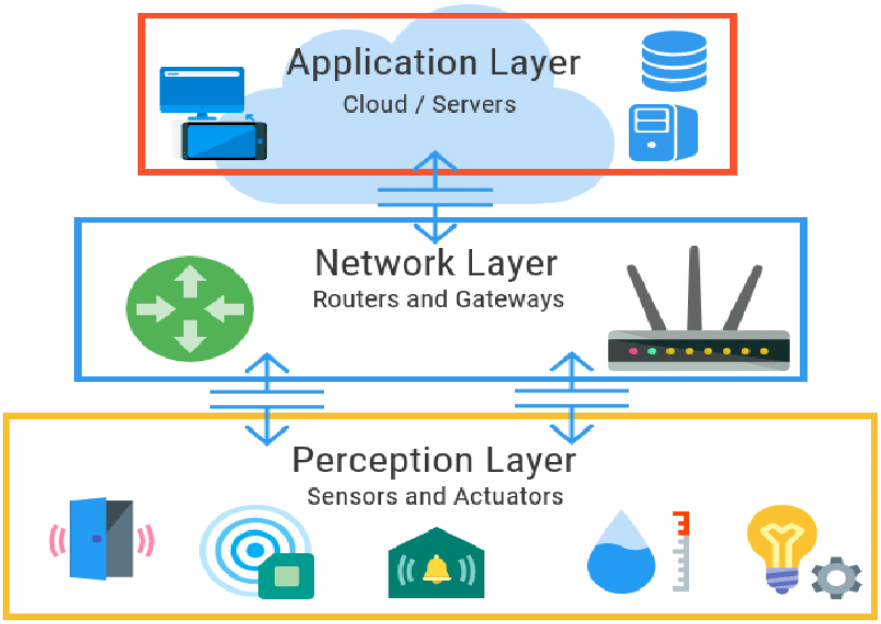


Fig. 1: Iot structure

*https://www.researchgate.net/profile/Qasem-AbuAlHaija/publication/347072341/figure/fig1/AS:970126849482752@1608307673848/IoT-Layered-Architecture-Considering-the-3-layer-Scheme-of-IoT-4.png

2.1 Perception layer

This is the architecture's initial layer, and it is responsible for gathering and pre-processing data with the ability to react to different events. Big data is created at this layer, where they are digitalized and sent to the Object Abstraction Layer for additional processing or storage through secure channels. The main technologies that make up this level are the following:

•**Sensors and actuators:** A crucial element of IoT systems is the sensors as they can observe the outside world and report the collected data so the machines or the people can use them. Sensors might be either analog or digital. Some examples of sensors that record their environment are temperature sensors, humidity sensors, pressure sensors, accelerometers, gyroscopes, cameras and many more. On the other hand, actuators are devices that can interact with their environment,some examples are step-motors, smart door lockers, smart light bulbs and more.

•**Objects and devices:** Devices or objects are various forms of hardware such as microcontrollers, household appliances smartwatches that can broadcast data over the internet and can be configured for certain purposes. These devices must have low consumption as many times they have to operate with batteries and have limited processing capabilities.

•**Transmission technologies:** These technologies are used for local network traffic or for transmitting the data to the gateway, their range can be from few centimeters to few kilometers. Some examples of such technologies are ZigBee, Bluetooth Low Energy, NFC, Wi-Fi.

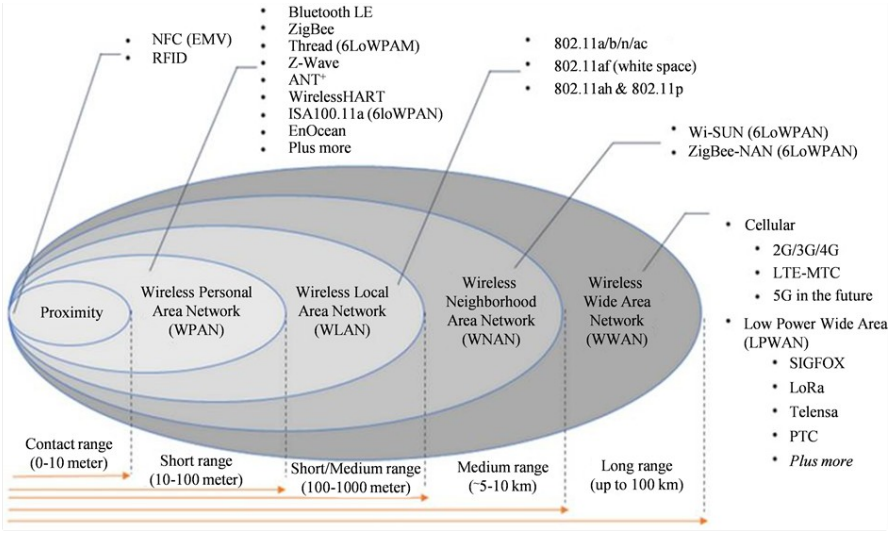


Fig. 2: *T*ransmission technologies

2.2 Network layer

This layer must handle large amounts of data generated by smart devices or gateways and send them via secure channels to the cloud provider that hosts the IoT application. The network layer uses a variety of network technologies to achieve scalability, large bandwidth, and security that various IoT applications need to function. To locate and route the data packets two main protocols are used in this layer, the IPv6 protocol is usually used by the gateways to connect with the outside world. But when a gateway is difficult to use like wireless sensor networks (WSN) a popular choice is IPv6 Low Power Wireless Personal Area Network (6LoWPAN). The objects are using technologies like Bluetooth, ZigBee, or NFC to reach the gateway.

2.3 Application layer

It receives massive amounts of data from the previous layer and stores, processes, and analyzes them. All the software and hardware required to provide a particular service over the internet are contained in the application layer on the cloud, more about the concerns and the abilities of the cloud in the next chapter.

3 Cloud

3.1 Cloud Computing

In both industrial and commercial applications, cloud computing has become an important paradigm that has garnered a lot of attention, especially in recent years. Without even knowing it, a lot of users and businesses utilize the cloud every day. For instance, all forms of email, online conferences, storage space for our personal files, or programs like Excel and Microsoft Word, that are not physically installed on our personal computer, are some forms of cloud computing applications. All the infrastructure to support these services exists somewhere in the world but customers might not be aware of the location of the servers that are hosting the source code of the programs they use for storing their data.

The term cloud computing is used to refer to computer resources offered on demand without the user's direct involvement by a worldwide network of connected servers accessible through the public internet [7]. These resources are mostly storage for data, processing power, and source code and must be delivered in seconds as if there was no difference if these applications were installed locally. This combination of computing resources can be used for many needs and many different kinds of applications.

To better understand how different organizations and ordinary people benefit from the cloud we must take a look at the three categories of services provided by the cloud.

•**Infrastructure as a service (IaaS)**: This is a service that offers basic processing and storage space over the internet. Instead of investing in their server or network infrastructure, businesses can rent those services and use them as needed instead of keeping those resources locally.

•**Platform as a service (PaaS)**: Developers can use the PaaS platforms for the creation of software applications. A full software environment can run at a service provider’s server while customers don't have to worry about the infostructure to support it. Google Cloud and Microsoft Azure are two examples.

•**Software as a service (SaaS):** Instead of purchasing a full license for an application, SaaS users can rent it for as much time as they need it through an Internet browser. SaaS follows a pay-per-use business model. These services are addressed to different customers.

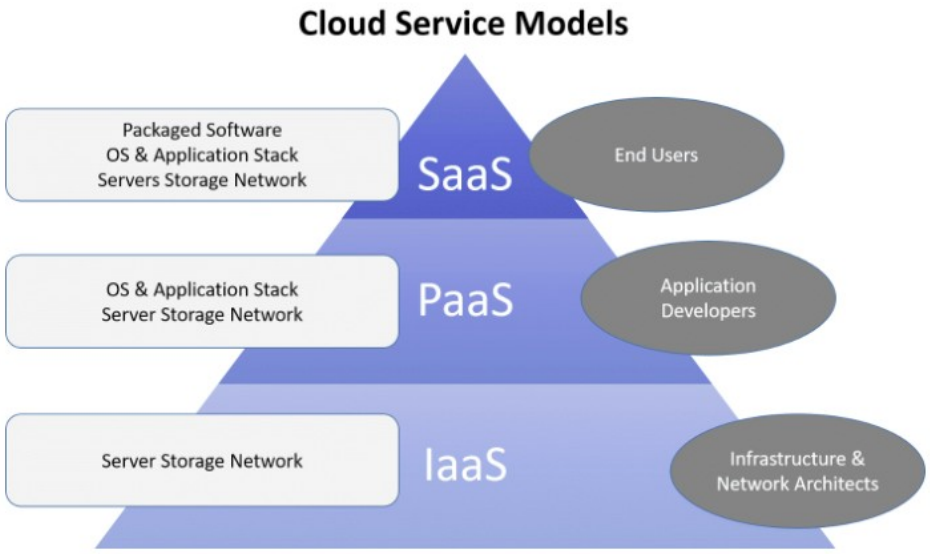


Fig. 3: Cloud Service Models *

*****source:** ***https://miro.medium.com/max/1400/1*OwcTPPaoQwE6e-cnOluGmw.png***

3.2 Cloud Computing vs Local Resources

There are many advantages of using cloud computing instead of our computer resources locally.

For companies that want to host an application online, the benefits of using a cloud provider are many. First of all, there is no need to invest in new hardware and software or maintain the existing ones because they are provided as pay-per-use much faster and cheaper. Companies can also quickly increase or reduce the amount of computing resources to much their needs. This is frequently done automatically giving the impression that resources are limitless and that applications can always handle the demand. Resources are returned to the resource pool when they are no longer required and they become available for others to use. We can also increase application speed by using cloud load balancing techniques in order to share these recourses more effi-

ciently among several applications that may run in the cloud. This means that by hosting our application in the cloud, the operational costs of many applications are reduced.

For the users using Cloud-Based Software instead of a desktop app, the main benefits are that they avoid the process of installing or updating the application locally and making backups of their files. They often avoid the cost of paying for software as it is often free with the display of ads or a small fee.

There are also many environmental benefits of cloud computing in addition to its technological and economical ones. On-site servers require ongoing electrical power and cooling systems to prevent overheating. One server can consume 500 to 1,200 watts of electricity per hour. It is estimated that 200 terawatt hours are used annually by larger data centers. That exceeds the whole national energy usage of several countries [27]. These numbers can be drastically reduced thanks to cloud computing. According to research from Northwestern University, the transferring of frequently used software applications to the cloud would reduce energy usage by up to 87 percent [24].

3.3 Cloud Computing Privacy Issues

To better understand what privacy is we must clarify the difference between privacy and security. Although these terms are related, data security and privacy are not the same. We can have only security without privacy but not the opposite. Data security enables us to defend our data against unauthorized access, it puts into practice the protocols that ensure our information's availability, confidentiality, and integrity [28]. On the other hand, privacy refers to an organization's responsibility to use people's personal information only for purposes to which they have given consent. These are usually personally identifiable information (PII), information that can be used to identify a specific individual such as financial data, medical records, social security/ID numbers, names, and birthdates. In many developed nations, it is protected by the constitution, making it a basic human right.

The nature of cloud services makes protecting customers' privacy a difficult task since the data are frequently being sent and maintained by a different party other than the data owner. The adoption of cloud services is significantly hampered by con-

cerns about the leakage of private information or the loss of sensitive data. These worries are indeed true. Services that depend on people's location, preferences, schedule, and social networks would need to take into account privacy since there are real potential dangers. For instance, it was revealed that in 2010 280 million user records containing personally identifiable information (PII) such as usernames, emails, phone numbers, and locations were exposed by insecure back-end databases of mobile apps [9]. Concluding, the public cloud is a popular architecture for cost reduction. However, depending on a cloud service provider to handle and store your data creates a lot of privacy problems. Using cloud services ultimately comes down to making trade-offs between security, privacy, costs, and advantages that it offers. Some concerns about privacy issues are described below [10]:

•**Lack of User Control:** When a SaaS environment is used, the service provider is in control of data storage, with limited visibility and control by the user. There is a risk of theft, leak, or misuse because the data of the customer are processed in the cloud by computers and software that they do not own or control.

•**Unauthorized Secondary Usage:** The service provider may make a profit from secondary uses of customers' data, mostly by targeted advertisement as part of the cloud computing standard business model. But some secondary data use might be unwanted to the data owner. For instance, a cloud provider is using the data for reasons other than those that were first agreed upon with the customer, like resale the costumer's data to other secondary businesses without his permission.

•**Data Proliferation and Transborder Data Flow:** Cloud providers can have many servers in different countries. Moving data for processing from one country to another increases risk factors such as legal complexity about how this data can be used.

•**Lack of Customer Trust:** Individuals will develop mistrust when it is unclear to them why their personal information is being asked, how it will be used, or by whom. This lack of control and visibility of the provider is also contributing to this mistrust.

3.4 Use of encryption in SaaS stages

If encryption is not used when it is required, the data may be exposed to external or internal threats. Despite the fact that utilizing encryption can help to retain data security on our infrastructure, putting encryption into practice is a difficult task and requires a lot of planning and design. It is difficult to determine when to encrypt the data and when decryption is necessary. The improper stage to encrypt data could result in poor computational resource management decreased performance, and increased costs. To better understand at which point the cryptography is most appropriate we have to look at the stages that the data went through from creation and sending to the server until processing.

•**Data in Transit:** Data sent between a server and a client, as well as between servers, can be encrypted by the network. This is done in order to prevent unauthorized users from eavesdropping on network traffic and to achieve the integrity of the data. Data must be transmitted through the Secure Sockets Layer when the device can be accessed via a web interface, and only by security protocols like Transport Layer Security [29].

•**Data at Rest:** This is when the data are stored on the server and are not in use. In SaaS environments, we have two options. The first option is, following the receipt of the data from the client, the cloud provider encrypts them with a key that he owns. This would secure the data from outside threats and allow the provider to retransmit or store the data in an encrypted form. The second option is, prior to uploading the data to the cloud, the client can encrypt it using an encryption key and an encryption method that only he knows. Given that the service provider lacks the decryption key, the SaaS application can only perform a restricted number of activities on the encrypted data. The encrypted data won't be readable by the SaaS application as a result he will not be able to process them.

•**Data in Use:** By using conventional encryption algorithms, encrypted data must first be decrypted before being loaded and processed in the server’s memory. This procedure should be done with care as the data may leak at this stage. This makes it difficult

to always maintain the data protected. Therefore, it will be very helpful to be able to process encrypted data without having to decrypt them. This can be done by using homomorphic encryption which we will analyze in the next chapter. This way the user can enhance their privacy from an honest-but-curious [30] cloud provider. As honest-but-curious, we referred to a cloud provider that adheres to the protocol correctly while also keeping a log of all its intermediate calculations for its personal profit.

4 Homomorphic encryption

4.1 Homomorphic encryption

Homomorphic encryption is a type of encryption that has the ability to do evaluations like additions and multiplications on encrypted data without having access to the secret key while the computation's outcome is always encrypted [20]. Generally speaking, homomorphism is a transformation that preserves structure between the plaintexts and the ciphertext. The necessity of using a secret homomorphism to encrypt the data of a bank is first mentioned in this paper [25] which shows that Time-shared computers needs manipulate the data without the need to first decrypt it.
Homomorphic Encryption works at the circuit level, which means that the functions to be used should only consist of binary operations such as AND and XOR. Homomorphic encryption is a term for an encryption scheme that can encrypt 0 and 1 and can multiply and add them. As we can see below, whatever calculation we perform on the encrypted numbers, must have the same result if we decrypt them or as if it was being decrypted.

$$E(m1 + m2) = E(m1) + E(m2)$$
$$E(m1 * m2) = E(m1) * E(m2)$$

Despite the fact that homomorphic evaluations can be performed on the encrypted data, it is crucial that this encryption must be as secure as the standard encryption methods. With the ability to compute encrypted data, it is a technology that we need to make use of as it has great potential in many real-world applications such as private statistical testing, private machine learning, and private neural networks [21],[22],[23]. Homomorphic encryption will play a key role in the future as more and more applications will rely on the cloud and people and companies are growing more apprehensive about the security and privacy of cloud data.

4.2 Homomorphic Encryption Types

Different homomorphic encryption techniques are in existence, some more powerful than others. They are distinguished by the types of functions that can be applied to the encrypted data. The homomorphic encryption types can be separated into three

generations of groups. Each generation tries to improve the problems that preceded them:

•**Partially homomorphic encryption:** Partially homomorphic encryption only allows a unlimeted number of operations to be evaluated on encrypted data, either just additions or just multiplications, and have only a small number of uses in real-life scenarios. Algorithms such as RSA and El Gamal belong to this category as they can only multiply encrypted data. These kinds of schemes are not safe to be used as homomorphic encryption schemes as they are vulnerable to CPA attacks.

•**Somewhat homomorphic encryption:** Allows computing additions on encrypted data as well as a limited number of multiplications. But as opposed to partially homomorphic encryption, a random element is included in the encryptions to prevent CPA attacks. This creates a new problem as the noise grows every time an evaluation is performed on the ciphertext, as a result, after a certain number of evaluations are applied to the ciphertext the correct decryption is not possible.

•**Fully homomorphic encryption:** In somewhat homomorphic encryption, after we have performed several arithmetic operations, the noise will have increased so much that it cannot be decrypted correctly. Fully homomorphic encryption includes bootstrapping techniques that allow the processing of the ciphertext further. This kind of homomorphic encryption enables much more if not unlimited multiplications and additions.

4.3 CKK Scheme

4.3.1 CKK Scheme

The Cheon-Kim-Kim-Song (CKKS) [11] is a fully homomorphic encryption scheme. The security of a CKKS system is based on the difficulty of learning with errors (LWE) over a ring of polynomial factors. CKK and all the other Fully homomorphic encryption (HE) became feasible after the development of a bootstrappable algorithm that could add and multiply homomorphically by Gentry [26]. Using CKKS, we are able to do calculations on complex value vectors and real values as well [8].

The CKK homomorphic encryption scheme supports the following four algorithms KeyGen, Enc, Dec, and Eval as any HE scheme. Let M and C represent, respectively, the plaintext and ciphertext spaces, a HE scheme Π = (KeyGen, Enc, Dec, Eval) consists of four algorithms and works as follows:

- KeyGen(1λ). The Key Generation Algorithm with respect to the security parameter λ, the output of this algorithm is a public-secret pair of keys pk and sk and an evaluation key evk.

- Encpk(m). The encryption algorithm encrypts a message m ∈ M into a ciphertext ct ∈ C using the public key pk.

- Decsk(ct). the decryption algorithm returns a message m ∈ M using the ciphertext ct and a secret key sk.

- Evalevk(f; ct1, . . ., ctk). Using the evaluation key evk and a tuple of ciphertexts (ct1, . . ., ctk) Eval carries out the function f on the ciphertext. The output of the evaluation algorithm is a ciphertext ct ∈ C.

4.4 Microsoft SEAL

The Simple Encrypted Arithmetic Library (SEAL) [33], is an open-source FHE library developed by Microsoft and it aims to make homomorphic encryption more accessible and has been used for industrial applications and academics. It was written in C++ and has no external dependencies. Two distinct homomorphic encryption schemes are included in Microsoft SEAL, the CKKS, and the BFV. For operations with encrypted integers, the BFV schemes can be used. While adding and multiplying encrypted real or complex numbers is possible only with the CKKS scheme, the downside of this scheme is that the results that yield are approximations of the result. CKKS is the best option in applications like adding encrypted real numbers, analyzing machine learning models on encrypted data, or calculating distances between encrypted locations. The BFV scheme is better suited for uses where precise values are required.

```
size_t poly_modulus_degree = 8192;
parms.set_poly_modulus_degree(poly_modulus_degree);
parms.set_coeff_modulus(CoeffModulus::Create(poly_modulus_degree, { 60, 40, 40, 60 }));
```

Fig. 4: Microsoft SEAL example

4.4.1 Using Microsoft SEAL CKKS

The whole process of encrypting text is described below (figure 5). As we can see there is an initial plaintext in the message m. This message must consist of floaing-point numbers. The first step is to encode the plaintext into a polynomial, and then it is encrypted with a public key. CKKS offers some operations that can be carried out on the message m once it has been encrypted into c, like addition, multiplication and rotation. After the encryption has been performed, we follow the reverse procedure to get the result. Below in this chapter, we will see separately each of the steps in detail to better understand the parameters and their effects on the security and performance of our computations.

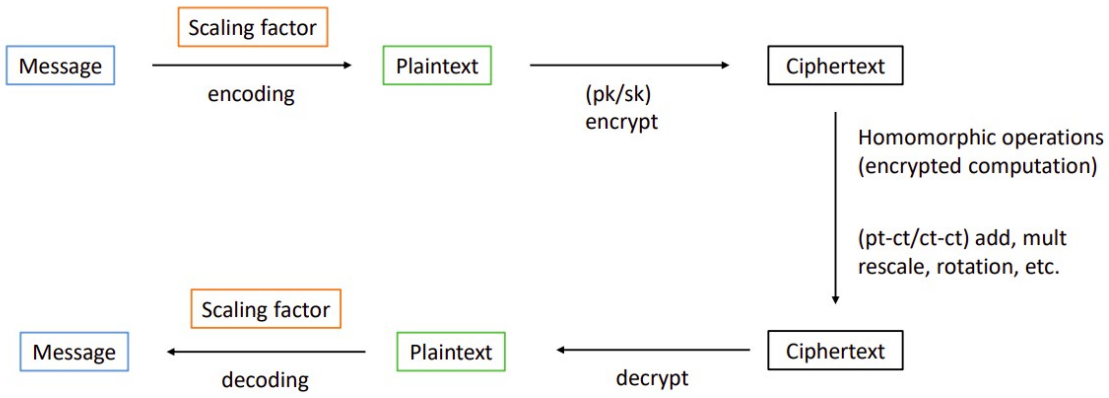


Fig. 5: The CKK Scheme*

*Source: https://yongsoosong.github.io/files/slides/intro_to_CKKS.pdf

4.4.2 Creating a message

The first step is the size selection of the message. This parameter is called polynomial modulo degree N and affects the number of slots of each ciphertext, this parameter is always a number of power of 2. Some common values that it takes are 2048, 4096, 8192, 16384, and 32768. The ciphertext has N/2 slots for floating point numbers (figure 6). We have to keep in mind that the message length is not the only reason to

select an appropriate polynomial modulo degree, a bigger polynomial modulo degree also means more computing capabilities like more multiplications as we will see later.

Num of slot	0	1	2	3	.....	N/2-5	N/2-4	N/2-3	N/2-2	N/2-1
Message	1.3	2.0	4.2	2.2	.....	6.3	2.4	1.4	2.3	3.3

Fig. 6: The message format

4.4.3 Encode of the message

The next step is to encode the message vector (figure 7). This is done in order to turn each real number in the vector into an integer. This is necessary since in the CKKS scheme the plaintext and the ciphertext elements are polynomials with integer coefficients. So, all the numbers in the vector we want to encrypt are multiplied by a scaling factor Δ. Bigger Δ means more precision. We can think of the scale as setting the bit-precision of our result. Example of fixed-point arithmetic with rescaling:

If we have two variables a = 0.331843, b = 2.226724 and scaling factor $\Delta = 10^6$ then:

a’ = a * Δ = 331843

b’ = b * Δ = 2226724

To calculate a + b, we perform the following steps.

1) calculate the result: c' = a' + b' = 2558567

2) rescale: c'/Δ = 2.558567

To calculate a * b, we perform the following steps.

1) calculate the result: c' = a' * b' = 738922772332

2) rescale: c'/Δ = 0.73892277233

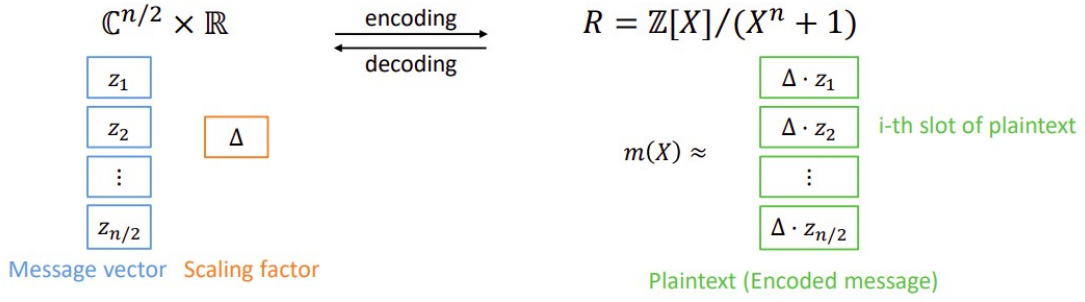


Fig. 7: Encode procedure

Instead of using decimal fixed-point arithmetic in CKKS, binary fixed-point arithmetic is utilized $\Delta = 2^p$. Additionally, CKKS operations add an additional approximation error that usually erases between 12 and 25 least significant bits so we have to keep that in mind as well when we set the encryption parameters.

4.4.4 Encryption of encoded massage

The encryption parameters must be set according to the number of multiplications and the decimal precision that we want. Each time a CKK ciphertext is multiplied, noise is added to the resulting ciphertext, on the other hand, encrypted addition and rotation do not have any significant impact on the ciphertext, so the design is relying on the multiplicative depth of our implementation. The multiplicative depth is determined by the length of its longest chain of consecutive multiplications. For example, the multiplicative depth of the $cp_0*cp_2*\ldots*cp_N$ is N, the multiplicative depth of the $cp_1*\ cp_2*\ldots*\ cp_N + cp_1*\ cp_2*\ldots*\ cp_M$ is M if M>N or N if N>M. It is advised to look for an implementation with the smallest depth possible because it will give us better performance.

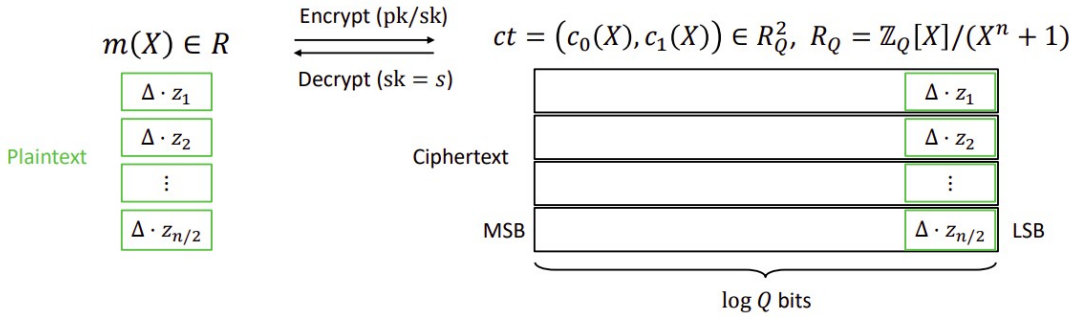


Fig. 8: Encryption procedure

*source: https://yongsoosong.github.io/files/slides/intro_to_CKKS.pdf

The encryption parameters are set by the coefficient modulus. The coefficient modulus Q (figure 8) is the sum of an array of sizes of primes in bits and we can think of it as space that our encoded float number will be, on our example code (figure 4) it

is {60,40,40,60} so it will be Q = 60*2+40*20 = 200 so our scale Δ needs to be much smaller in order to “fit” (figure 6). The position of each number in the coefficient modulus array is important. There are the outer primes (the first and last numbers) and the inner primes which are all the rest. We must keep in mind the following things considering the coefficient modulus:

- The number of inner primes determines the multiplicative depth limit. In our example (figure 4) we have a multiplicative depth of two.
- Poorer performance, longer run-time of the computation, and bigger memory consumption is associated with larger size of primes.
- The precision both before and after the decimal point is determined by prime sizes. For example, if we want precision to be 10 bits, we must set the inner primes at 25 and the outer primes are of size 35.
- The polynomial modulus degree limits the coefficient modulo bit count (figure 9). In our example, if we set the polynomial modulus degree to the value 8192 and we have Q = 60*2+40*20 = 200 and the max is 218.

poly_modulus_degree	max coeff_modulus bit count
1024	27
2048	54
4096	109
8192	218
16384	438
32768	881

Fig. 9: Polynomial modulus degree and max coeff modulus bit

4.4.5 Multiplication & Rescale

Scales in ciphertexts increase as a result of encrypted multiplication. The ciphertext's scale shouldn't be too close to the coefficient modulus's overall size, or else there won't be enough space in the ciphertext to store the scaled-up plaintext (figure 8).

Reducing the scale and stabilizing scale expansion is made possible by the rescale functionality of the CKKS scheme. Rescaling is a modulus switch operation. An inner prime in coefficient modulus is removed as part of the modulus switching process, as a result, the ciphertext is scaled down by the removed prime (figure 9). For example, if the ciphertext's scale is Δ and its current coefficient modulus is P as its last prime, rescaling to the next level converts the scale to Δ/P and eliminates the prime P from the coefficient modulus. We must choose carefully primes sizes for the coefficient modulus because we want to have complete control over how the scales are changed.

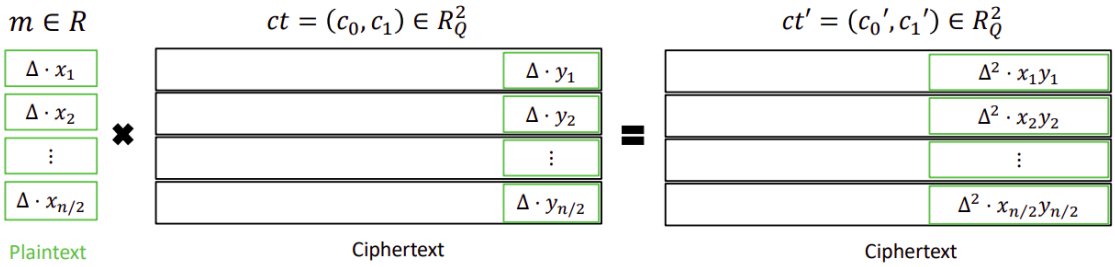


Fig. 10: Encrypted multiplication

***source:** https://yongsoosong.github.io/files/slides/intro_to_CKKS.pdf*

The number of inner primes restricts the multiplicative depth as we described above and limits the number of rescalings that can be performed. Every time we rescale, we go down a level by removing an inner prime. When there is no inner prime left then we have reached level zero. On level zero we can't apply any more evaluations on the encrypted message and we must decrypt it.

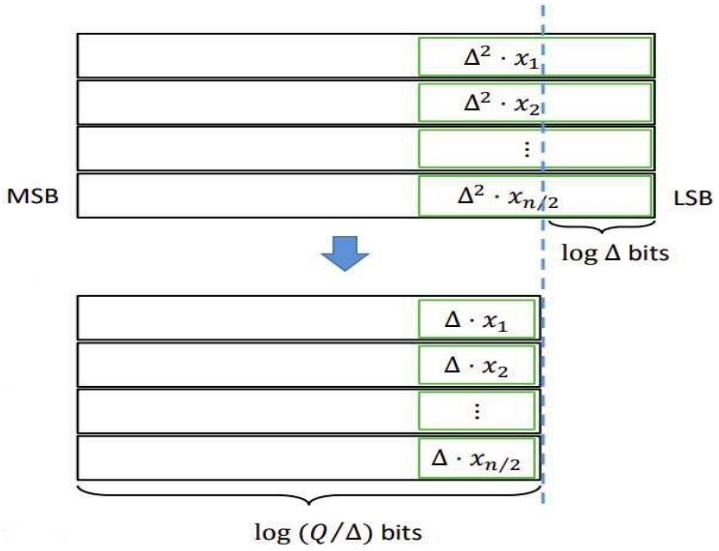


Fig. 11: Rescale operation
source: https://yongsoosong.github.io/files/slides/intro_to_CKKS.pdf

5 Pattern recognition

Pattern recognition originates from engineering and statistics. Pattern recognition is a classification process and its objective is to help machines to make automated decisions based on data given. A pattern recognition classifier needs to be capable of adapting and learning from examples. Applications for pattern recognition include image processing, speech and fingerprint recognition, etc. The application domain should always be taken into account while designing a pattern recognition system. Pattern recognition algorithms are typically divided into two categories: supervised and unsupervised. Most of the time, labeled data for training is used to train a pattern recognition system. This is known as supervised methods. Other algorithms can be employed to find previously unidentified patterns when labeled data are not available. This is known as unsupervised methods. To design such systems, we must follow the three steps that are presented below.

•**Pre-processing:** Its purpose is to make a more consistent set of data by minimizing variances. Pattern discovery during the training phase is more challenging if there are a lot of redundant, irrelevant information, or noisy data. Cleaning, instance selection, normalization, one hot encoding are a few examples of data preprocessing.

•**Feature extraction:** In this step, we must discover distinctive characteristics that are common across several data classes as this will allow us to make better predictions with fewer data. There must be little intraclass variation. The features extracted and the feature extraction techniques depend on the application.

•**Classification:** The system must recognize the patterns of the inputs and assigns each one to the proper class using the characteristics that were retrieved from the learning processes. In the literature, there are two different kinds of learning processes supervised and unsupervised learning.

5.1 Perceptrons

5.1.1 Perceptron

Perceptron is a pattern recognition algorithm and consists of one neuron (figure 12). A neuron has a number of inputs and the result of this equation is passed to an activation function in order to decide whether or not that neuron should activate. When categorizing data that can be separated linearly, the perceptron is a very useful technique. But as it was discovered with the XOR problem, they run into serious limitations with data sets that do not follow this pattern.

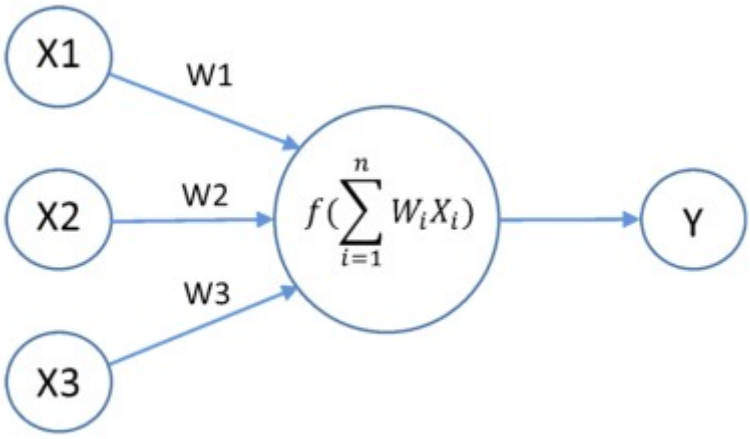


Fig. 12: A Perceptron

5.1.2 Multi-Layer Perceptron

Multi-Layer Perceptrons correct this problem that is present in the single perceptrons. Perceptrons are the basic unit of Multilayer Perceptrons. The Multi-Layer Perceptrons are a powerful tool for categorizing groups of data that cannot be linearly separated. This is accomplished by adding a hidden layer and an output layer as we figure 13.

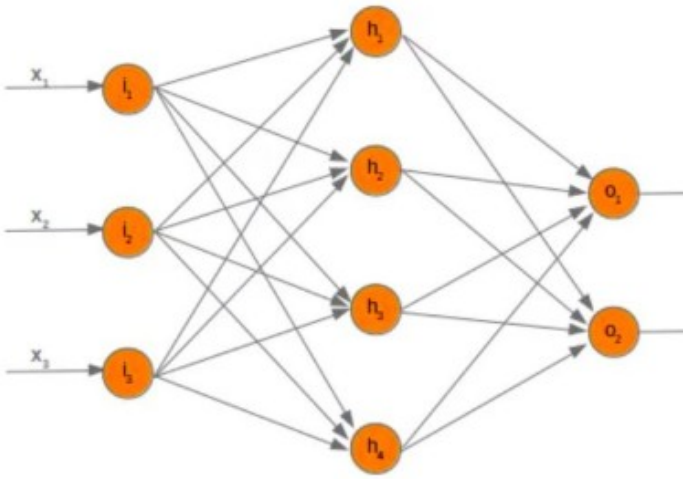


Fig. 13: Multilayer Perceptron

The MLP algorithm is as follows:

1.The first step is to calculate the output of the hidden layer. To do this, we simply calculate the product of the matrix of weights with the input matrix figure 14.

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Fig. 14: First layer calculations

2.On every output from the first step, MLPs apply an activation function. There are numerous activation functions that we can use, including tanh, sigmoid Function and ReLU.

3.The results taken by the operations from the previous step are multiplied by the table of weights of the output layer (figure 15).

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} wh_{11} & wh_{12} & wh_{13} & wh_{14} \\ wh_{21} & wh_{22} & wh_{23} & wh_{24} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

Fig. 15: Second layer calculations

4.Finally, we have two options for what we can do with this result, in the case of training we use the output to correct the weights, and in the case of testing, we use the output to make a decision based on the data we used.

6 Related work

Modern embedded systems, which cover a wide range of applications and different communication protocols, are commonly resource constrained, making it difficult to deploy the traditional software-based techniques used for detecting and containing malicious software in general-purpose computing systems [40][41]. An important concern for the Internet of Things based systems is to ensure trusted execution of applications and communication inside vehicles, factories, smart buildings, for well-being, and healthcare [43][44][45]. Even more challenging is implementing verification checks of applications executing in low-performance and limited memory embedded systems [42].

From the above papers mentioned on traditional security techniques we can conclude that there is a tradeoff between the edge-device's computation burden and the security of the application, as the closer to the edge-device a security feature is implemented, the more secure but also more complex the application will be. This is also true for our case but fully homomorphic encryption is even more resource demanding on both memory and computation but it promises great potential for our privacy [47].

In some early attempts to use homomorphic encryption on IoT, there have been proposed several homomorphic schemes other than fully homomorphic encryption that are more lightweight for uses cases like IoT and mobile applications, but those schemes can apply only certain kinds of queries as they are on based partial homomorphic encryption or somewhat homomorphic encryption, for instance [48][49] they propose a partially homomorphic encryption scheme based on ElGamal encryption that supports only homomorphic addition so they can do only certain kinds of queries like the summation of some values or to find the range of those values, but the operations of the scheme are light enough to run on embedded system or mobile phone. In more recent years more and more attempts are made to utilize fully homomorphic encryption on the edge, for example, they have implemented a scenario [51] where homomorphically encrypted data from automobiles are collected from the cloud but still, the encryption is not implemented on the edge device.

The present thesis used the SEAL-Embedded library to perform fully homomorphic encryption completely on the edge and measure the computational load and time for such use cases, and in addition, it combined encrypted inference

techniques based on [36][37][38] in order to have a complete use case scenario where the user can perform queries from his phone from the data that the board encrypted.

7 Implementation

7.1 System Overview

We can summarize the system we implemented in its three main functions as we can see in figure 16 which we will analyze further below.

1.Initially, the embedded system will encrypt and send via NFC to the android the encrypted values. The mobile will play the role of the gateway and will send the data to the server for storage.

2.From another application, the user will be able to ask the server to perform actions on the data stored on the server.

3.The server is responsible to retrieve the right information, processing the data and answer to the user's query.

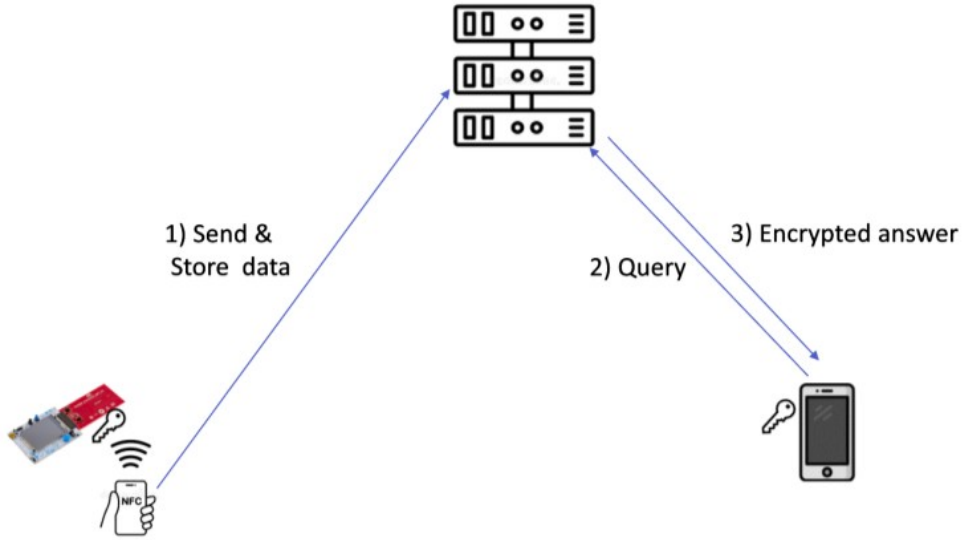


Fig. 16: System overview

7.2 Edge to server side

The whole process that we designed in order to send data homomorphically encrypted to the server from the edge device is described below in figure 17. The data are values from the sensors collected by the embedded system. The data initially will exist in plaintext form stored in the board's memory, this is done because as we will see later in chapter 7.3 homomorphically encrypted data takes up much more memory space than plaintext data, so the board will encrypt the data piece by piece. Each piece of data will be deleted as soon as the user collects them.

When the user wants to collect the data to offload board memory must do it through the mobile application that was developed. When the collection process is complete and all the ciphertext is on the mobile phone, then the ciphertext will be send and stored in a non-relational database on the server along with its metadata through TCP. In this chapter we will deal with:

- The connection of the embedded system with a NFC antenna and the installation of the appropriate libraries.
- The installation of encryption library on the embedded system.
- Development of the android application that will collect the data and send them to the server.

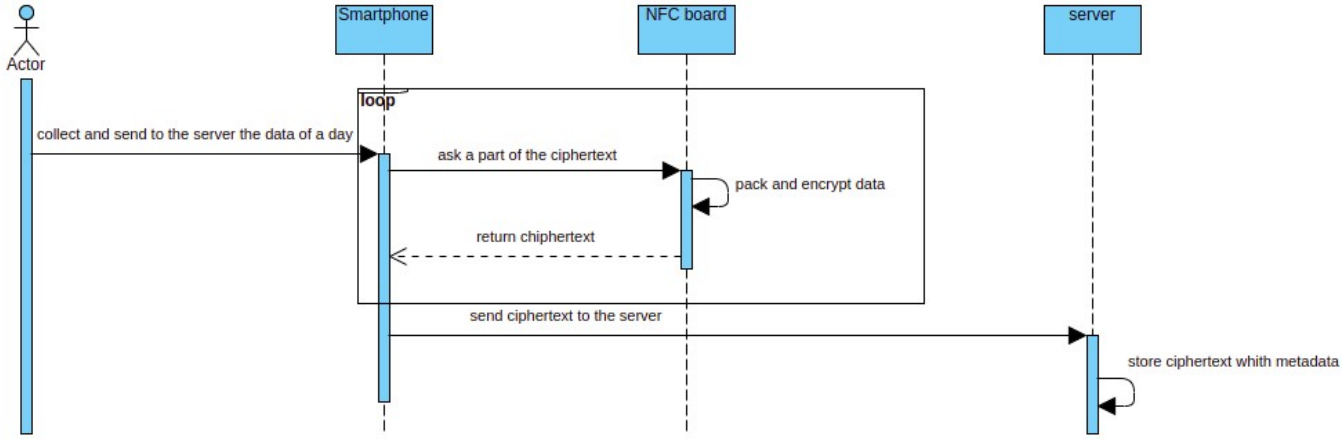


Fig. 17: Sequence diagram

7.2.1 Implementation

7.2.1.1 Embedded system

The 32F746GDISCOVERY Discovery kit [31] used in this project contains the STM32F746NG microcontroller, which is based on an Arm Cortex-M7 core. The audio, multi-sensor support, graphics, security, video, and high-speed connectivity features of the Discovery kit can enable a wide range of applications. The ARDUINO connectivity support offers lot of expansion potential with a wide range of add-on boards.

Fig. 18: 32F746GDISCOVERY Discovery kit

Main features of the board:

1.STM32F746NGH6 Arm® Cortex® core-based microcontroller with 1 Mbyte of Flash memory and 340 Kbytes of RAM, in BGA216 package

2.4.3” RGB 480×272 color LCD-TFT with capacitive touch screen

3.Ethernet compliant with IEEE-802.3-2002

4.ARDUINO Uno V3 expansion connectors

7.2.1.2 I2C protocol

The Inter-Integrated Circuit (I2C) bus is a two-wire serial interface protocol which is used in consumer products and was created by the Phillips Corporation [32]. The I2C protocol allows communication between numerous devices connected to the same bus network (figure 19).

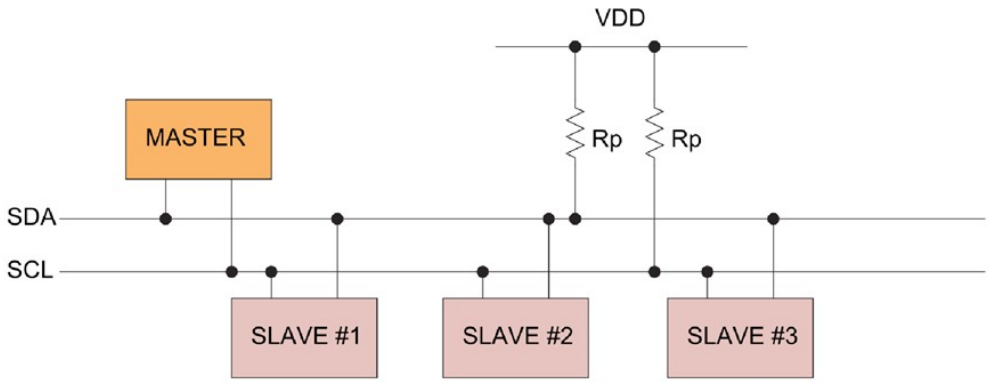


Fig. 19: I2C bus*
***Source:** https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html*

All data in I2C are carried over two bidirectional lines, one line for the data which is called the serial data line (SDA), and another one serial clock line (SCL) for the clock whose purpose is to synchronize the nodes. The I2C bus is connected to a common power supply (VDD) so when it is idle it is in HIGH state, if a node wants to transmit then must toggle the lines by pulling LOW with its open drain pins.

I2C follows a master/slave hierarchy, there is one device the master that controls the clock of the bus (thus the communication speed), addresses slaves, and writes to or reads data from slave nodes. The start signals and the stop signal are represented by the negative edge and positive edge of SDA respectively when the SCL is in HIGH state (figure 20). The slaves on the other hand are nodes that only answer when invoked by the master via their specific address. Therefore, it is essential to prevent address duplication among slaves. A data transfer is never started by a slave. The speeds we can write/read data are 400 kbits/s which is called The Fast Mode transfer rate, compared to the standard data transfer rate of 100 kbits/s.

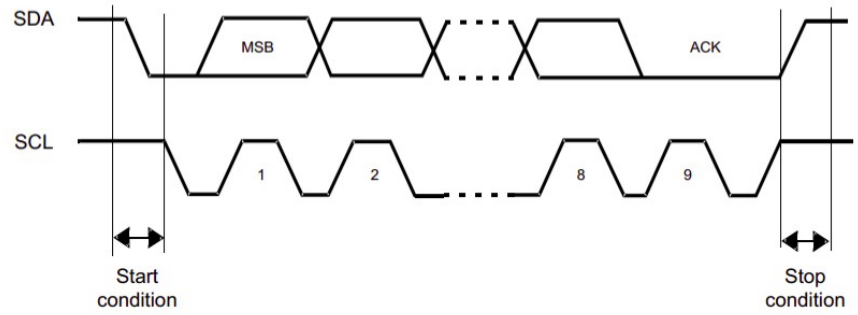


Fig. 20: I2C waveform

Every I2C message must have a slave address, the operation that the master wants to execute, and the data packets that are 8-bit long (figure 21). Only a read or a write operation occurs during a transmission session.

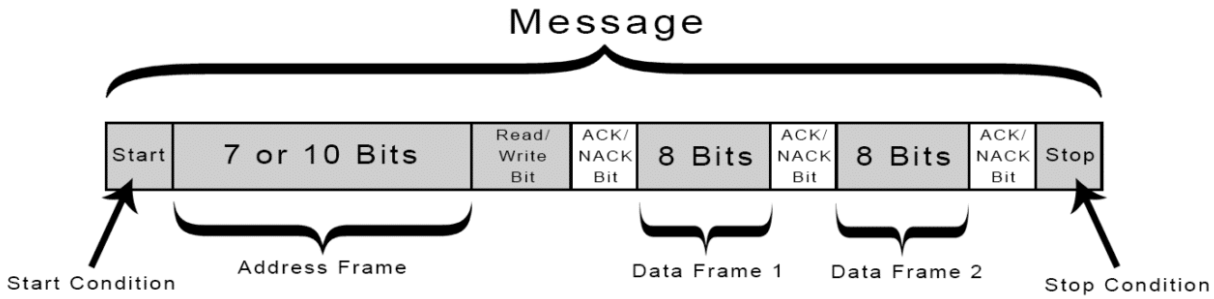


Fig. 21: I2C message frame
***Source:** https://www.circuitbasics.com/wp-content/uploads/2016/01/Introduction-to-I2C-Message-Frame-and-Bit-2.png*

7.2.1.3 NFC Antenna

The NFC antenna used for the project is a Class 5 antenna daughter card ST25DV Discovery ANTC5 by STMicroelectronics [35] which includes (figure 22):

- 40 mm x 24 mm, 13.56 MHz inductive antenna etched on the PCB
- ST25DV04K Dynamic NFC / RFID tag
- I2C interface connectors
- Energy harvesting output (VOUT) with a 10 nF capacitance filtering circuit
- GPO configurable as RF WIP/BUSY output, to indicate that an RF operation is ongoing

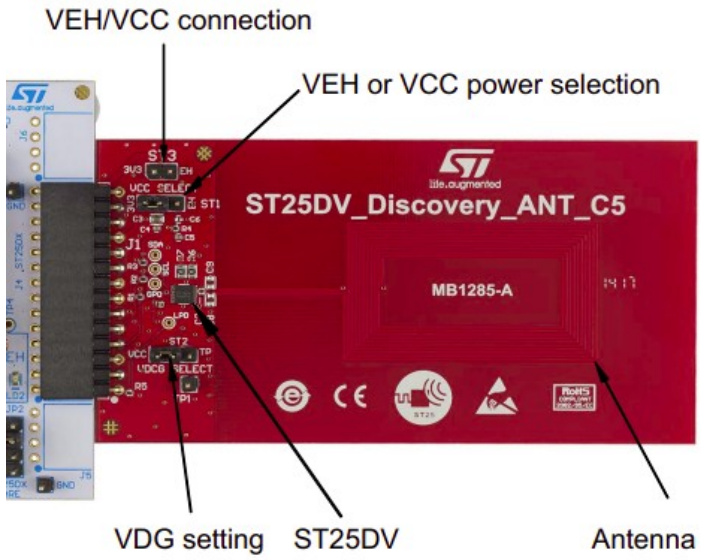


Fig. 22: NFC antenna

We choose this antenna because is embedded with the ST25DV64KC (figure 22) and we can make use of its fast transfer mode capabilities to send the ciphertext from the board to the phone via NFC. The antenna has three features that we make use of:

- The mailbox that the embedded library uses to transfer messages between the RF and I2C worlds. The data in this mailbox can hold up to 256 Bytes.

- A GPO output that can trigger an interrupt when multiple RF events like field change, memory write, activity, Fast Transfer end, user set/reset/pulse and I2C events like memory write completed, RF switch off occurs.

- The MB_CTRL_Dyn dynamic register contains activity indicators about the mailbox and the fast transfer mode state.

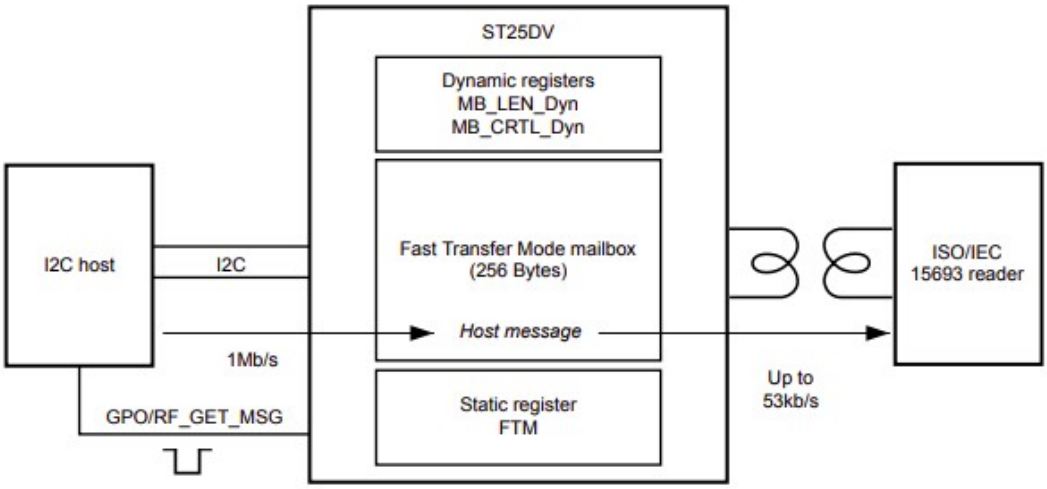


Fig. 23: ST25DV64KC I2C device

In order to send data from the I2C host to the RF reader Fast transfer mode must be enabled, the mailbox must be empty, VCC power must be present, and the I2C host must first write the message which contains the data. To see if there is an I2C message in the mailbox, the RF device must poll the MB CTRL Dyn register. After the RF has finished reading the entire message, the mailbox is once again considered free and is capable to receiving new messages. A GPO interrupt will alert the I2C host that the message has been read by the RF.

7.2.1.4 ST25 fast transfer mode embedded library

The embedded library ST25 fast transfer mode (ST25FTM) enables quick data transfer between an NFC reader and a dynamic tag. The ST25FTM protocol was created to control this kind of data transfer using the ST25DVxxKC chip family while allowing for error detection and recovery and using the least amount of meta-data possible. ST25FTM is based on a state machine to keep track of the data sent or received (figure 24). The ST25FTM library is a middle-ware and was made independent of the microcontroller and the dynamic tag. This means that a lower-layer API for the middle-ware needs to be implemented for the intended hardware.

ST25FTM_WRITE_IDLE	When the ST25FTM_SendCommand function is called to provide data to be transmitted, the FTM transmission state machine initializes all the required state variables and the CRC IP.
ST25FTM_WRITE_CMD	The ST25FTM transmission state machine prepares the next segment to be sent, according to the acknowledgment scheme selected (doing nothing/computing segment CRC).
ST25FTM_WRITE_SEGMENT	The ST25FTM transmission state machine prepares the next message to be sent, according to its position in the segment.
ST25FTM_WRITE_PKT	The ST25FTM transmission state machine actually writes the message to the FTM memory.
ST25FTM_WRITE_WAIT_READ	The ST25FTM transmission state machine waits until the message has been read by the peer device.
ST25FTM_WRITE_READ_ACK	In case an ACK is expected from peer device, the ST25FTM transmission state machine poll to read the ACK message.
ST25FTM_WRITE_DONE	The transmission has successfully gone to its end.
ST25FTM_WRITE_ERROR	An unrecoverable error occurred.

Fig. 24: ST25FTM transmission states

7.2.1.5 Software implementation

STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. It is based on the Eclipse framework and GCC toolchain for the development, and GDB for the debugging. It allows the integration of the hundreds of existing plugins that complete the features of the Eclipse IDE.

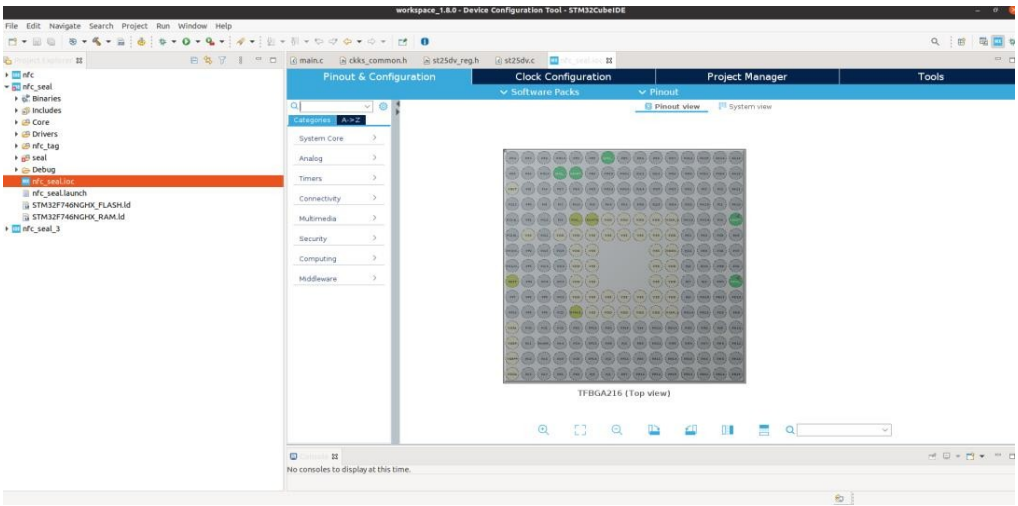


Fig. 25: STM32CubeIDE

At this point, we have two hardware devices, the ST25DV_Discovery_ANT_C5 and the 32F746GDISCOVERY Discovery board, and the ST25FTM software to put together. ST25FTM library needs the NFC drivers to use the tag since was developed independently from the NFC tag, and the NFC tag drivers need the I2C drivers of STM32F746NGH6 microcontroller since the driver was developed independently from the board, so we will set up the project from the bottom to the top starting with the I2C drivers. We can initialize I2C drivers for our board with the following parameters using the STM32CubeIDE (figure 26).

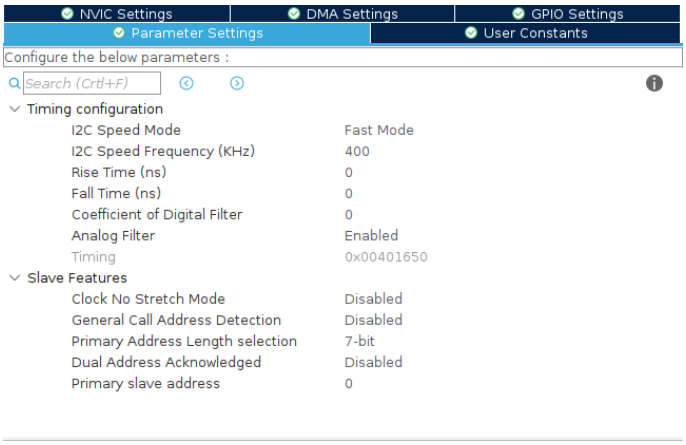


Fig. 26: I2C set up

Then we can copy the ST25DV drivers and the ST25FTM library into the project. The ST25DV driver's files (figure 27 blue square) include all the addresses for writing and reading from the I2C peripheral and a standard API to use the tag as recommended by the STM32 Cube methodology, they are also compatible with the HAL drivers so we will have no problem using them together with I2C drivers we create earlier. The API is used by the fast transfer mode middleware libraries (figure 27 red square) to write to the mailbox.

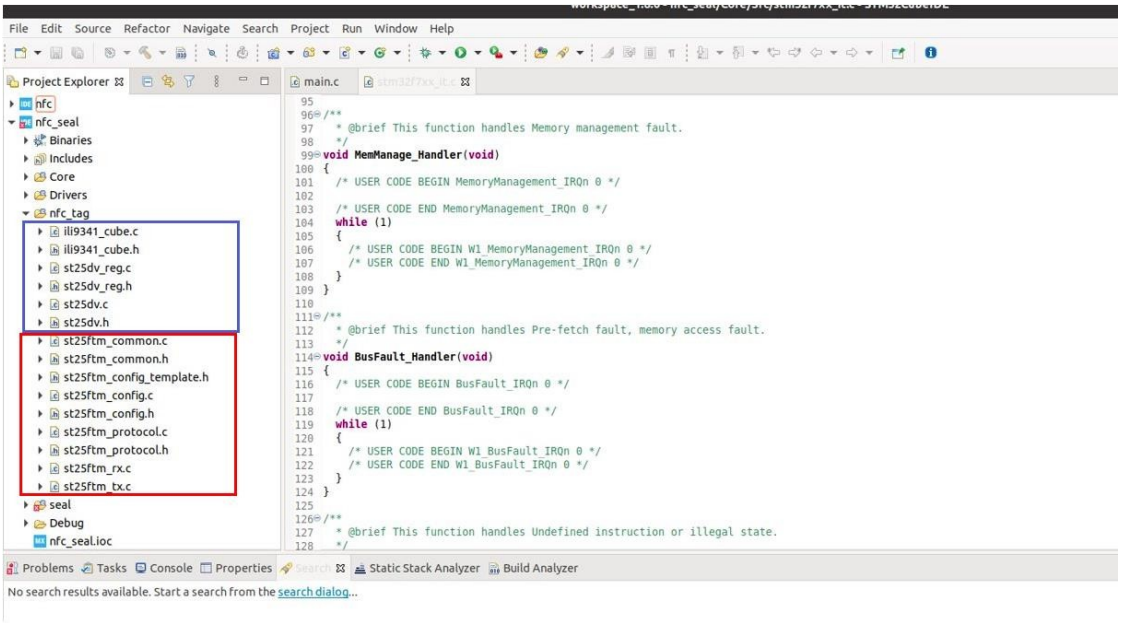


Fig. 27: Project setup

Now that we can write to the mailbox, we need to also to install the interrupt for the library to know for different events that occurred. So once again we enable interrupts from STM32CubeIDE, every time an interrupt occurs the interrupt handler assigns the value 1 into the global variable GPO_Activate so the FTM library knows an event occurred and must check the MB_CTRL_Dyn register.

```
static void ManageGPO( void )
{
  uint8_t itstatus;

  if(GPO_Activated == 1)
  {
    GPO_Activated = 0;

    ST25_RETRY(BSP_NFCTAG_ReadITSTStatus_Dyn(BSP_NFCTAG_INSTANCE, &itstatus));

    if((itstatus & ST25DV_ITSTS_DYN_FIELDFALLING_MASK) == ST25DV_ITSTS_DYN_FIELDFALLING_MASK)
    {
      FieldOffEvt = 1;
    }

    if((itstatus & ST25DV_ITSTS_DYN_FIELDRISING_MASK) == ST25DV_ITSTS_DYN_FIELDRISING_MASK)
    {
      FieldOnEvt = 1;
    }

    if((itstatus & ST25DV_ITSTS_DYN_RFPUTMSG_MASK) == ST25DV_ITSTS_DYN_RFPUTMSG_MASK)
    {

      mailboxStatus = ST25FTM_MESSAGE_PEER;
    }

    if((itstatus & ST25DV_ITSTS_DYN_RFGETMSG_MASK) == ST25DV_ITSTS_DYN_RFGETMSG_MASK)
    {
      mailboxStatus = ST25FTM_MESSAGE_EMPTY;
    }
  }
}
```

Fig. 28: Usage of GPO_Activated

In figure 29 we can see the connections between the NFC antenna and the board. The antenna provides a power supply for the I2C bus to work. We chose the pins according to Appendix A.

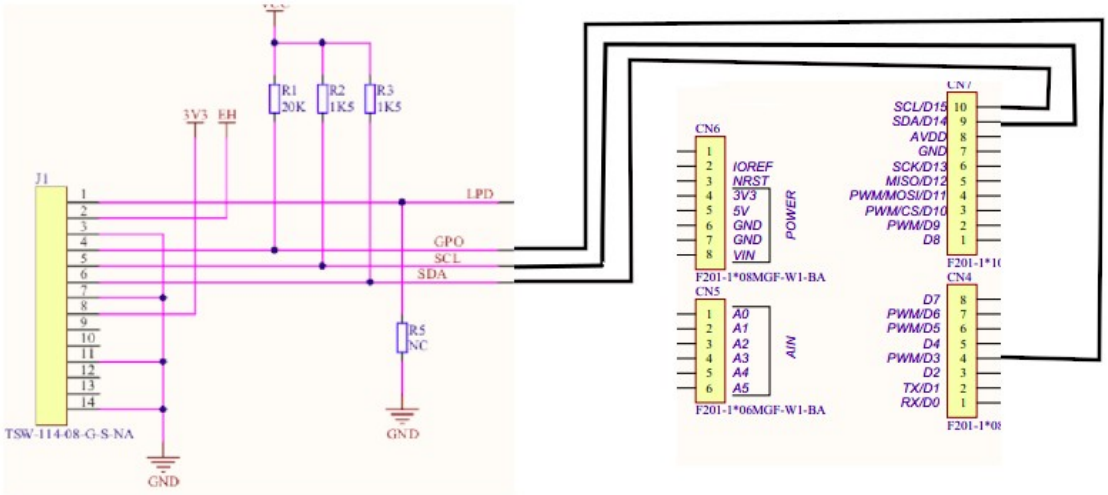


Fig. 29: Board-antenna connection

7.2.1.6 Microsoft SEAL IoT library

The SEAL-Embedded is the first homomorphic encryption library designed for embedded devices that can support the CKKS approximate homomorphic encryption scheme. The library consists of two pieces, the library that the board is running and the adapter server module that transforms data encrypted by SEAL-Embedded into a format compatible with the Microsoft SEAL library. Additionally, the SEAL-Embedded adapter has functions for generating the public and secret keys in a format that the SEAL-Embedded library can understand. To be able for the board to encrypt homomorphically, the creators of the library have implemented some optimizations to lower the memory consumption of homomorphic encryption encoding and encryption while maintaining high performance. The main optimizations are:

•**RNS partitioning** is the first significant way that the library overcomes memory constraints. By operating to a single prime at any given point, the isomorphism established by the Chinese remainder theorem [39] preserves arithmetic in Rq0 Rq 1 ⋯Rq L−1 with arithmetic in subrings Rqi that can be performed independently. SEAL- Embedded exploits this property by performing encryption in one subring Rqi at a time, allocating just enough memory to carry out operations for a single prime component.

•**Data type compression**, memory compression of some polynomials is the second method SEAL-Embedded uses to lower memory usage. SEAL-Embedded take advantage of the secret key polynomial s ternary structure or polynomial u in the asymmetric case and use only two bits per coefficient to store its values. Additionally, SEAL-Embedded uses compression to the error sampling's output. Due to the fact that each error sample is an integer between [21, 21], only 6 bits are needed to represent each sample.

•**Memory pooling**, utilizing a memory pool is the third significant way SEAL-Embedded overcomes memory constraints. SEAL-Embedded carefully calculated the minimum amount of memory needed for the library to run effectively in both

symmetric and asymmetric cases. These sizes are used by SEAL-Embedded to allocate all the memory required up front for a specific library configuration.

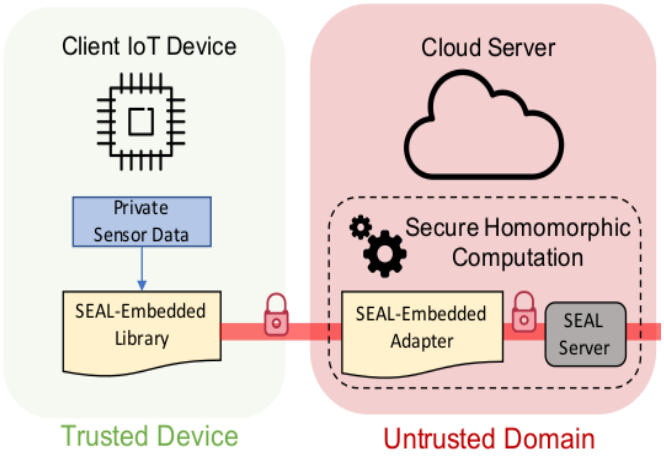


Fig. 30: SEAL-Embedded

7.2.1.7 Android application

For the android application part, a modified version of the STSW-ST25001 provided by STMicroelectronics was used. The STSW-ST25001 makes use of the STSW-ST25SDK001 - ST25 software development kit which is a library that can write to NFC and is compatible with fast transfer mode to collect the ciphertext. The modifications were to collect all six parts of the ciphertext. The ciphertext consists of six parts since the primes numbers we chose were six as we explain on the RNS partitioning in the previous chapter.The board calculates each part and sends them to the phone one by one.

Fig. 31: Android application

7.2.1.8 Power and time measurements

In figure 32 the first image is showing the time that each part of the ciphertext takes to be encrypted and sent to the phone through NFC. As we can see the sending time remains stable in all ciphertext parts and takes about 35 seconds but that depends on the distance and the position of the phone from the NFC antenna, each ciphertext piece is 65.54 KB. On the other hand, the encrypting time is increasing steadily for each ciphertext part.

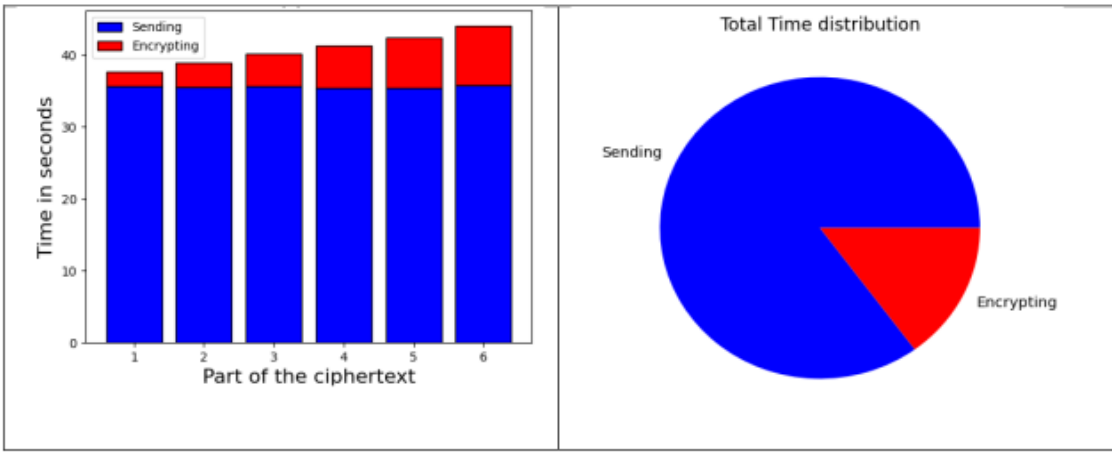


Fig. 32: Time measurements

In Figure 33 we can see the power consumption of the embedded system during the encryption and when the board is sending the ciphertext. We can see a slight increase when the board is encprypting.

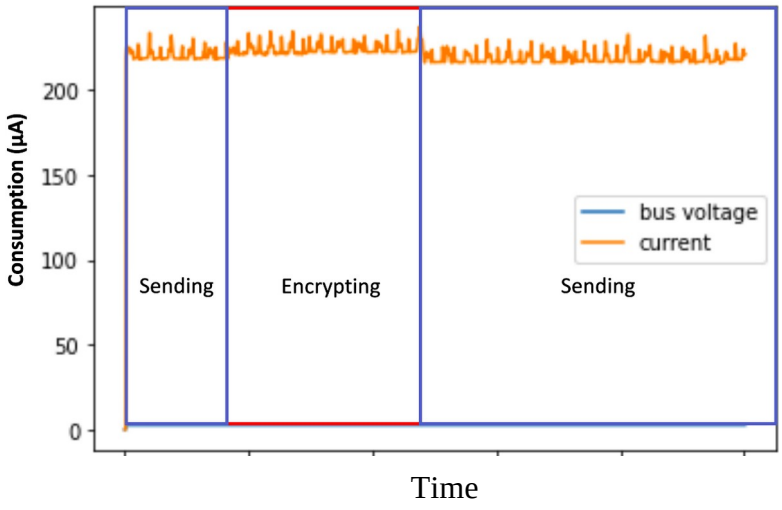


Fig. 33: Power measurements

7.3 Server to client side

In this chapter, we will explain the whole process and the design of how the user can send a query to the server and receive the encrypted answer based on the data stored on the server. To demonstrate the abilities of the system we design a real-world scenario which we will explain inmore detail later.

The front end of the application consists of a mobile application that is responsible to send a query to the server through TCP, receiving/decrypting the ciphertext, and displaying on the mobile phone the results that the user requested. On the back end, the scenario is an encrypted inference of a perceptron classifier that classifies the values encrypted by the embedded system.

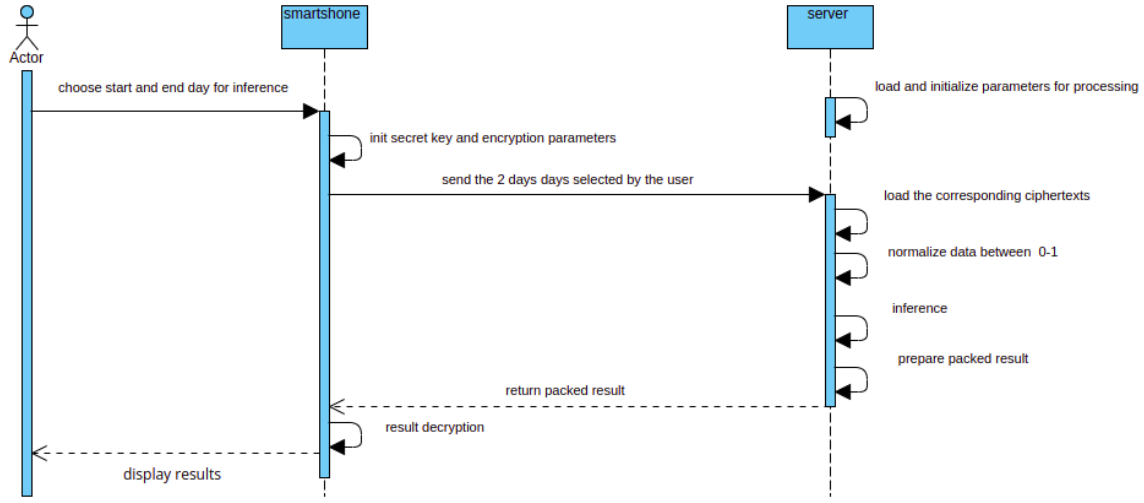


Fig. 34: Sequence diagram

The whole process for doing that is described below (figure 34).

- The first thing that both server and user do is to load a predetermined set of parameters that are agreed upon. These parameters are created by the client using the adapter of the Microsoft SEAL embedded and are mandatory to make possible the rotation and other evaluations. The client creates the SEAL evaluator and other objects that the server requires and sends them to the server while withholding the secret key, this way the server can't decrypt the ciphertexts but can process them.

- The next step for the server is to load from memory the encrypted values that the user requested. The server knows which encrypted text belongs to which data and what the user requested but does not know the contents inside as they are encrypted.

- The values that the board encrypts are the original ones that were recorded by the board, but in order to pass through the neural network, they must be normalized between the range zero and one on the server. Although it would be convenient if the board send the normalized values directly to the server because it would save us from encrypted evaluations, it would be difficult to use the values for some other applications that need the original values like calculating the average or extracting other statistics.

- The server runs the inference network created and trained in TensorFlow. The weights exist in plaintext form on the server. The data processing consists of two matrix multiplications between the encrypted values send by the board and the plaintext weights extracted from the model we implemented.

- And finally, the server prepares and sends the encrypted answer to his query and sends it to the user.

7.3.1 Implementation

7.3.1.1 Dataset description

A suitable dataset to use to build our system and to test our use case is the Room Occupancy Estimation Data Set [34]. This dataset contains measurements from environmental sensors to estimate the presents of people in a room. These measurements are temperature, light, sound, CO2, and digital passive infrared (PIR sensor). The measurements were taken from one 6m x 4.6m room and the dataset can be used to measure the space-frequency rate and occupancy rate which helps us to determine whether and how space is being used and help us to make decisions about the future [53]. Below in figure 35, we visualize the data to see if there is any correlation between the data and to have a better idea of the dataset in general. The

dataset has sixteen attributes four of them are the temperature, four are light density, four are sound classes one is CO2 slope and two are PIR. The dataset's classes(num of occupants fig. 35) are four and they are the number of occupants and take values from 0 to 3. In this work, we merge the classes from 1 to 3 as we wanted only to find presence in the room at a given time.

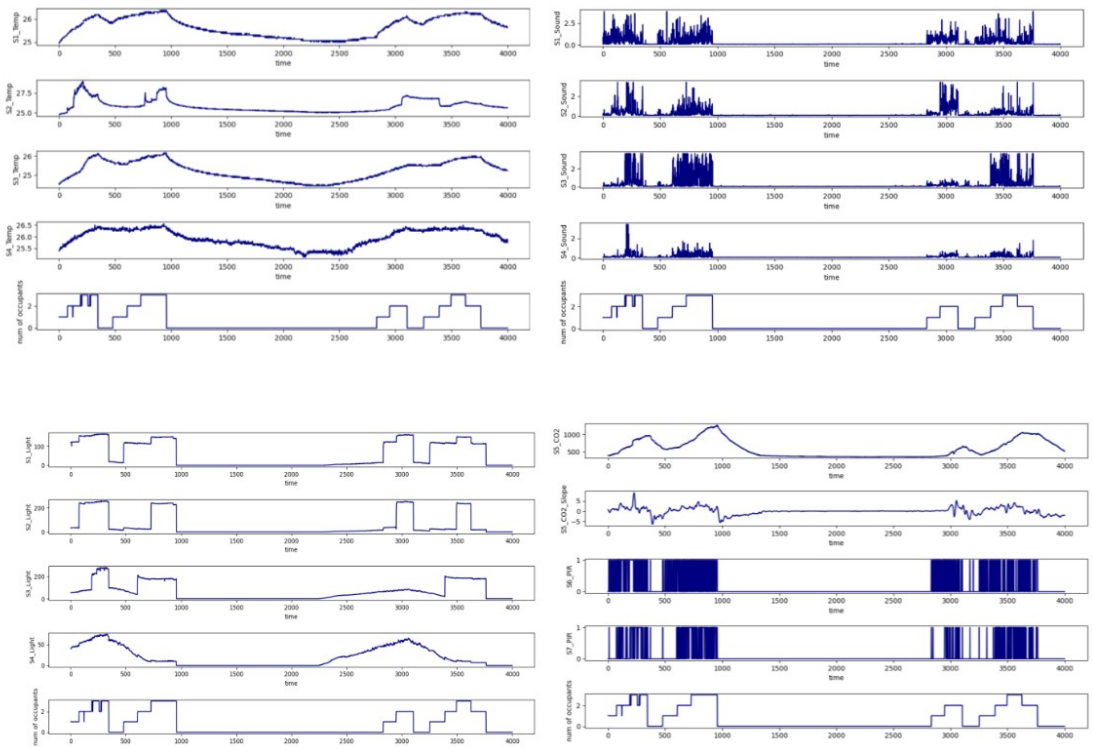


Fig. 35: Dataset visualization

From the above plots in figure 35, we conclude that we can separate the dataset's attributes into two categories. The data that can provide a very good indication of how many people are in the room but it takes them some time to rise or to fall into the category that they belong, those values are CO2 readings and temperature, and the data that can provide a more immediate indication like light, sound, and PIR but we can't rely always on them as they are not always present, for

example, someone who is in the room may have the lights closed or make no sound.So, we will use all the attributes and let the neural network find a pattern for us.

7.3.1.2 Perceptron architecture & training

We designed the architecture of the model keeping in mind that the vast majority of its calculations will have to run in the encrypted domain, so a couple of things that we have to keep in mind are the multiplication depth that is restricted and the fact that only multiplication and addition are allowed and it is impossible to use division or comparison. This means that we can't make use of the most common activation functions since they make use of that kind of calculation, for example, the sigmoid function use division and relu, and step functions use comparison so we must avoid them. In figure 36 we can see the architecture of the neural network that we trained. It takes as input all the 16 attributes and has square activation function in the middle, the square activation function is the simplest non-linear function that we can use as it consists of just one multiplication. For better training results we used the sigmoid activation function only on the output layer of the neural network. Although we explained why it is impossible to use the sigmoid function at the intermediate layers of the neural network, we can still use it only on the last layer. This is because the sigmoid function is a monotonically increasing function. The application of a monotonically increasing function will not change the result of the neural network, the smaller number will stay smaller and the bigger number will stay bigger, so we can skip the application of the sigmoid function, and just choose the biggest of the two outputs of the FC-2 layer to be our inferred result. So, we use the sigmoid function to train the plaintext network but we are not using it on the encrypted inference.

Layer
FC-1
Square activation function
FC-2
Sigmoid activation function

Fig. 36: Neural network architecture

```
for layer in model.layers: print(layer.get_config(), layer.get_weights())
```

```
{'name': 'flatten', 'trainable': True, 'batch_input_shape': (None, 16, 1), 'dtype': 'float32', 'data_format': 'channels_last'} []
{'name': 'dense', 'trainable': True, 'dtype': 'float32', 'units': 12, 'activation': {'class_name': 'Activation', 'config': {'name': 'activation', 'train
       0.611,  0.234, -0.59 ,  0.649],
       [ 0.17 ,  0.585, -0.075,  0.589, -0.202, -0.065,  0.249,  0.274,
         0.353, -0.24 , -0.257,  0.716],
       [-0.686,  0.799,  0.966,  0.334, -0.967,  0.612,  0.179,  0.616,
        -0.297, -0.429, -0.213,  0.425],
       [-0.091, -0.161,  0.13 ,  0.08 ,  0.229,  0.025, -0.975, -0.176,
         1.266,  0.654, -0.05 , -0.354],
       [-0.29 ,  1.082,  0.794,  1.03 , -0.859,  0.892,  0.642,  1.189,
        -0.536, -0.777, -0.634,  0.715],
       [-0.611, -0.106,  0.713,  0.778,  0.005, -0.002,  0.303,  0.501,
         0.06 , -0.183, -0.501,  0.788],
       [-0.177, -0.01 ,  0.082,  0.016, -0.271,  0.064, -0.135, -0.073,
         0.361, -0.417, -0.234, -0.05 ],
       [-0.112,  0.355, -0.086, -0.007,  0.141, -0.059, -0.203,  0.034,
         0.401, -0.199,  0.176,  0.114],
       [-0.655,  0.776, -0.002,  0.226, -0.291,  0.51 ,  0.496,  0.281,
        -0.077, -0.399, -0.954,  1.308],
       [-0.376,  0.449,  0.723,  0.506, -0.319,  0.209,  0.465,  0.337,
        -0.602, -0.253, -1.184,  0.632],
       [-0.238,  0.297,  0.006,  0.518, -0.133,  0.454,  0.285,  0.183,
         0.144,  0.066, -0.18 ,  0.52 ],
       [-0.163,  0.669, -0.079,  0.212, -0.28 , -0.188, -0.675,  0.384,
        -0.148,  0.309,  0.302,  0.445],
       [-0.118,  0.03 ,  0.506, -0.085,  0.045,  0.049, -0.102, -0.215,
         0.687,  0.427, -0.514,  1.04 ],
       [-0.928,  1.167,  0.769,  1.014, -0.851,  0.455, -0.264,  0.579,
        -0.305,  0.441,  0.146, -0.4  ],
       [-0.358,  0.281,  0.137,  0.003, -0.314,  0.484, -0.218,  0.187,
         0.078,  0.025,  0.072,  0.691],
       [ 0.037,  0.145,  0.309, -0.442, -0.354,  0.371,  0.076, -0.199,
        -0.163, -0.175, -0.433,  0.514]], dtype=float32), array([-0.099,  0.078,  0.168,  0.052, -0.234,  0.19 , -0.895,  0.25 ,
        0.866,  1.   ,  0.625, -0.844], dtype=float32)]
{'name': 'dense_1', 'trainable': True, 'dtype': 'float32', 'units': 2, 'activation': 'sigmoid', 'use_bias': True, 'kernel_initializer': {'class_name': '
       [-1.074,  0.175],
       [-0.878,  0.819],
       [-0.828,  0.501],
       [-0.87 ,  0.227],
       [-0.808, -0.278],
       [ 0.717, -0.895],
       [-0.825,  0.47 ]]
```

Fig. 37: Extracted weights matrices

In figure 38 we can see the accuracy of the perceptron we trained using the test dataset. The A is the human present class and the B is the no-present class.

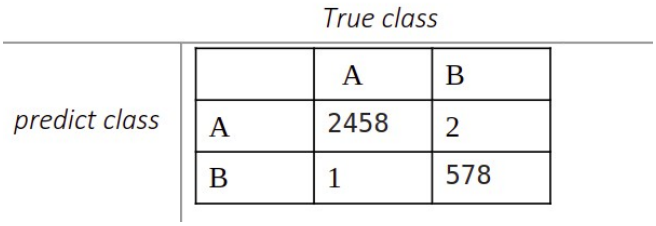


Fig. 38: Confusion matrix

7.3.1.3 Encrypted matrix multiplication implementation

To implement the encrypted perceptron, we need a matrix multiplication algorithm. A simple method for multiplying two matrices one encrypted and one ciphertext of size N ×N is to use N distinct ciphertexts, one ciphertext for each value using only one slot, although this solution may work for certain use cases it is impossible to do this in our implementation because of the many extra data that the board will have to send to the server. For example, each ciphertext of degree 8192 is 440KB, if we wanted to send 4096 values to the server using only one slot per ciphertext then we have to send 1760MB instead of 440KB. So, our goal is to try to use all the slots of each ciphertext that the board encrypts and sends to the server, so no empty slots arrive at the server. For that purpose, it was desing a matrix multiplication algorithm that supports ciphertext packing technique, so we can encrypt multiple values into a single ciphertext and use them more efficiently.

The matrix multiplication algorithm 1 takes as an input an encrypted matrix of dimension NxP and a plaintext matrix of dimension PxN and outputs a ciphertext with the encrypted result. The algorithm also takes as an input a matrix in column-major format order (figure 39) and yields a matrix that has the same format the rest blue slots are ignored. This way, we can have many matrix multiplications in a row with the same function, as many of our chain multiplication depths allow us. The cp.A and the pt.B represent the encrypted matrix and the plaintext matrix respectively, and the U is a binary mask in the size of each row,n and p are the dimensions of the first encrypted matrix and m is the number of columns of the second plaintext matrix.An illustrative explanation of the algorithm can be found in Appendix B and the code for the SEAL implementation can be found in appendix C.

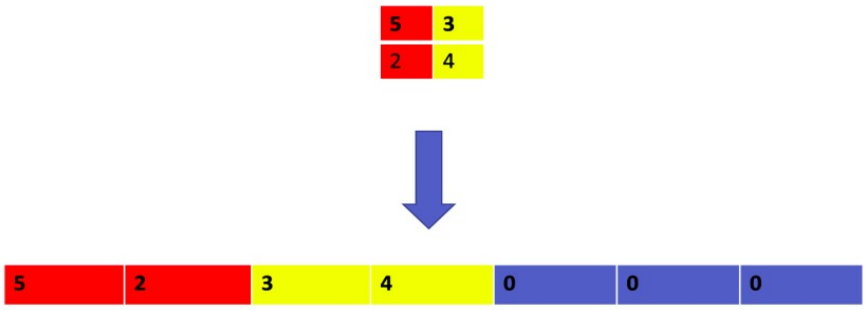


Fig. 39: Matrix column-major format

Algorithm 1

- procedure MatMult_ciphertext_plaintext(ct.A, pt.B,n,p,m)
- [STEP 1]
- For k=0 to p do
  - Ct.A= Rot(ct.A;n)
  - Ct.Temp[k] = Mult(ct.A;U)
- End for
- [STEP 2]
- For k=0 to p do
  - For g =0 to m do
    - Ct.B = Ct.Temp[k]
    - Ct.Temp[k] = Add(Rot(Ct.B;g*n), Ct.Temp[k])
  - End for
- End for
- [STEP 3]
- For k=0 to p do
  - For g =0 to m do
    - Pt.Temp[k] = pt.B[k*n+g]
  - End for
- End for
- [STEP 4]
- For k=0 to p do
  - ct.AB = Add(Mult(Pt.Temp[k], Ct.Temp[k]) , ct.AB)
- End for
- return ct.AB

7.3.1.4 Encrypted inference

In figure 40 we can see the ciphertext that the board is encrypting. The ciphertext's polynomial modulo degree is 8192 which means that the cipher has 4096 slots, in these slots we can fit 256 matrices of size 1x16. For simplicity, we chose to collect values every 6 minutes so that one ciphertext corresponds to exactly a day's measurements.

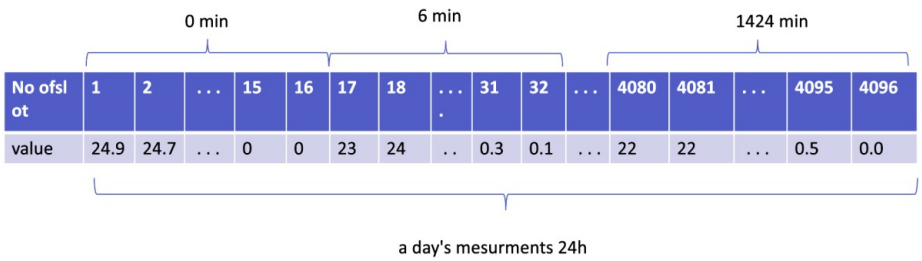


Fig. 40: Ciphertext encrypted by the board

In figure 41 we can see the exact implementation of the encrypted inference. The matrices with red letters are encrypted and the ones with black letters are plaintext. In addition, the gray part is executed on the server while the orange one is sent to the final user as the answer. As we can see the user does not apply the sigmoid function as we explain in chapter 7.3.1.2. The code for the SEAL implementation can be found in appendix D.

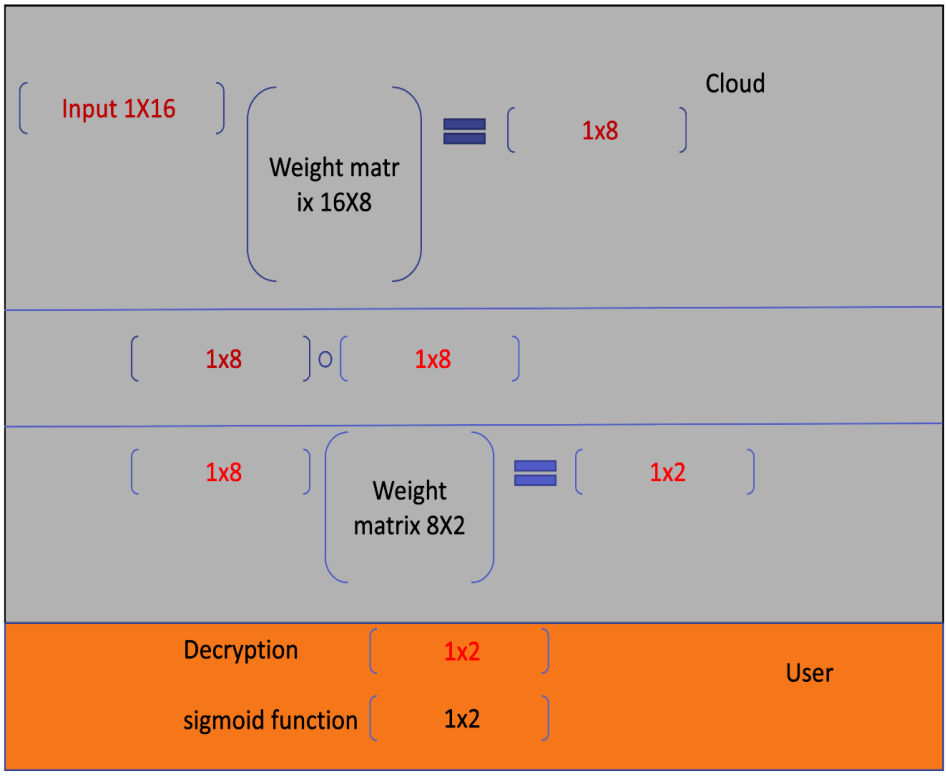


Fig. 41: Encrypted inference

In figure 42 the first image is the output of the plaintext model from chapter 7.3.1.2 and the second one is the output of the encrypted model from chapter 7.3.1.4 for the same input. As we can see we have lost some precision after the third-fourth decimal number but this is expected, as we explained in the chapter 3.3.1 CKK is an approximate homomorphic encryption scheme, but this is not a problem as two decimal numbers are more than enough for the application.

Fig. 42: Plaintext-ciphertext comparison

In figure 43 we can see in detail the initial parameters of our encryption, the scale as it grows on each layer and the rescales operation, we apply to keep the scale under control. When the calculations are on the second layer, we have finished with the encrypted inference but we keep applying to ciphertext rescale operations because each time we remove a prime the ciphertext size is also reduced so the server will have to send less data to the user.

Encryption parameters :
poly_modulus_degree: 8192
coeff_modulus size: 218 (30 + 30 + 30 + 30 + 30 + 30 + 38) bits
scale 25

	scale	Modulus chain index	Size of ciphertext
Initial ciphertext	25	5	440 KB
First layer	75	5	
Rescale	45	4	
Activation function	90	4	
Second layer	140	4	
Rescale	110	3	
Rescale	80	2	140KB
Rescale	50	1	
Rescale	20	0	76KB

Fig. 43: Encrypted neural network analysis

7.3.1.5 Mobile application

Because there is no available library for android, all of the FHE functionality of the application was created in native C++, we built a dynamic library *.so compatible with the phone's architecture and wrapped it with JNI interface so we can call it through the android app. As we described earlier, we used a polynomial modulo degree of 8192 which means we have 4096 available slots, the neural network has 2 outputs one for every class so in each ciphertext we can fit 2048 inferences. In figure 44 we have sent a request to the server to evaluate 10 instances of the dataset, and the server

classified 10 of them in class B and 0 in class A, the unknown class is the remaining slots that arrived empty.

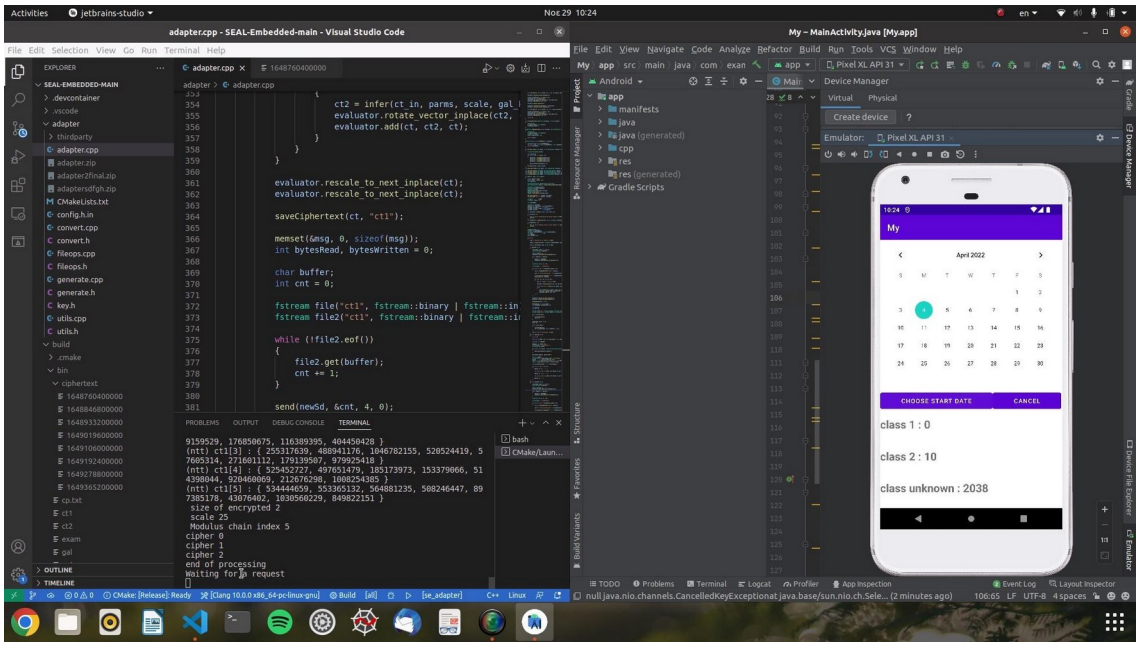


Fig. 44: Android client application

8 Conclusions and Future Work

The highlights of this thesis were the use and testing of the SEAL-Embedded on the 32F746GDISCOVERY Discovery kit, its combination with encrypted inference techniques found in literature, and the development of the mobile application that makes the query to the server.

More precisely on the embedded system side, we ran the library and tried to send the ciphertext through NFC. Although the NFC is not the transmission technology with the highest throughput we manage to send the cipher text in a reasonable time around 3.5 minutes in total for both encryption and transmission which is acceptable for some use cases but in some cases, it might be better to use Bluetooth or ZigBee. On the server side, a multilayer perceptron was trained based on a dataset we found and use some techniques to execute it on the encrypted domain and also we create an android app for the user to manage the data.

There are lots of aspects of this work that can be improved. For instance, we manage to protect the privacy of the user as we used homomorphic encryption, but the integrity of the data is not protected by any means, since evaluation can be done on the encrypted data, some malicious users may change the content of the ciphertext, so some other types of encryption can utilized like hash functions and symmetric encryption like AES to make sure that only the server can process the ciphertext.

Also, we can take action to protect the secret key that exists on the board and the code itself. As some papers show there is a chance for an attacker to find the secret key with a side channel attack [52]. We can also try to run the librarie to a trust execution environment but this will be challenging since the TEE has limed resources.

9 Bibliography

[1] D Kaur, A., Singh, V. P., & Singh Gill, S. (2018). The Future of CloudComputing:Opportunities, Challenges and Research Trends. 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social,Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference On.doi:10.1109/i-smac.2018.8653731

[2] Dores, C., Reis, L. P., & Lopes, N. V. (2014). Internet of things and cloud computing. 2014 9th Iberian Conference on Information Systems and Technologies(CISTI). doi:10.1109/cisti.2014.6877071

[3] Choudhury, T., Gupta, A., Pradhan, S., Kumar, P., & Rathore, Y. S. (2017). Privacy and Security of Cloud-Based Internet of Things (IoT). 2017 3rd International Conference on Computational Intelligence and Networks (CINE). doi:10.1109/cine.2017.28

[4] Hamza, R.; Hassan, A.; Ali,A.; Bashir, M.B.; Alqhtani, S.;Tawfeeg, T.M.; Yousif, A.TowardsSecure Big Data Analysis via FullyHomomorphic EncryptionAlgorithms.Entropy 2022, 24, 519.https://doi.org/10.3390/e24040519

[5] Peralta, G.; Cid-Fuentes, R.G.; Bilbao, J.; Crespo, P.M. Homomorphic Encryption and Network Coding in IoT Architectures: Advantages and Future Challenges. Electronics 2019, 8, 827. https://doi.org/10.3390/electronics8080827

[6] Natarajan, D., & Dai, W. (2021). SEAL-Embedded: A Homomorphic Encryption Library for the Internet of Things. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(3), 756–779. https://doi.org/10.46586/tches.v2021.i3.756-779

[7] Surbiryala, J., & Rong, C. (2019). Cloud Computing: History and Overview. 2019 IEEE Cloud Summit. doi:10.1109/cloudsummit47114.2019.00007

[8] Cheon, J.H., Kim, A., Kim, M., Song, Y. (2017). Homomorphic Encryption for

Arithmetic of Approximate Numbers. In: Takagi, T., Peyrin, T. (eds) Advances in Cryptology – ASIACRYPT 2017. ASIACRYPT 2017. Lecture Notes in ComputerScience(), vol 10624. Springer, Cham. https://doi.org/10.1007/978-3-319-70694-8_15

[9] Zuo, C., Lin, Z., & Zhang, Y. (2019). Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps. 2019 IEEE Symposium on Security and Privacy (SP). doi:10.1109/sp.2019.00009

[10] Pearson, S., & Benameur, A. (2010). Privacy, Security and Trust Issues Arising from Cloud Computing. 2010 IEEE Second International Conference on Cloud Computing Technology and Science. doi:10.1109/cloudcom.2010.66

[11] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, ASIACRYPT 2017, Part I, volume 10624 of LNCS, pages 409–437. Springer, Heidelberg, December 2017

[12] https://www.st.com/en/evaluation-tools/32f746gdiscovery.html

[13] Babu, S. M., Lakshmi, A. J., & Rao, B. T. (2015). A study on cloud based Internet of Things: CloudIoT. 2015 Global Conference on Communication Technologies (GCCT). doi:10.1109/gcct.2015.7342624

[14] Navani, D., Jain, S., & Nehra, M. S. (2017). The Internet of Things (IoT): A Study of Architectural Elements. 2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS). doi:10.1109/sitis.2017.83

[15] https://www.forbes.com/sites/raufarif/2021/06/05/with-an-economic-potential-of-11-trillion-internet-of-things-is-here-to-revolutionize-globaleconomy/?sh=18ad816e5f29

[16] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and publickey cryptosystems,” Commun. ACM, vol. 21, no. 2, pp. 120–126, 1978.

[17] C. Gentry, “Fully Homomorphic Encryption Using Ideal Lattices,” in Proceedings of the Annual ACM Symposium on Theory of Computing, 2009, vol. 9, pp. 169–178.

[18] T. Sander, A. Young, and M. Young, “Non – Interactive CryptoComputing for NC1,” FOCS Comput., pp. 554–557, 1999.

[19] C. Gentry, “Fully Homomorphic Encryption Using Ideal Lattices,” in Proceedings ofthe Annual ACM Symposium on Theory of Computing, 2009, vol. 9, pp. 169–178.

[20] N. N. Kucherov, M. A. Deryabin and M. G. Babenko, "Homomorphic Encryption Methods Review," 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2020, pp. 370-373, doi:10.1109/EIConRus49466.2020.9039110.

[21] A. Dalvi, A. Jain, S. Moradiya, R. Nirmal, J. Sanghavi and I. Siddavatam, "Securing Neural Networks Using Homomorphic Encryption," 2021 International Conference on Intelligent Technologies (CONIT), 2021, pp. 1-7, doi:10.1109/CONIT51480.2021.9498376.

[22] Zebbiche, K. & Khelifi, Fouad & Bouridane, Ahmed. (2008). An Efficient Watermarking Technique for the Protection of Fingerprint Images. EURASIP J.Information Security. 2008. 10.1155/2008/918601.

[23] J. Kim and A. Yun, "Secure Fully Homomorphic Authenticated Encryption," in IEEE Access, vol. 9, pp. 107279-107297, 2021, doi:10.1109/ACCESS.2021.3100852.

[24] https://earth5r.org/environmental-benefits-cloud-computing/

[25] Ronald L. Rivest Len Adleman Michael L. Dertouzos “ON DATA BANKS AND PRIVACY HOMOMORPHISMS” Massachusetts Institute of Technology Cambridge, Massachusetts 1978

[26] C. Gentry and D. Boneh, A fully homomorphic encryption scheme, 09. Stanford University Stanford, 2009, vol. 20.

[27] https://www.nature.com/articles/d41586-018-06610-y

[28] A. Durcikova and M. E. Jennex, "Introduction to Confidentiality, Integrity, and Availability of Knowledge, Innovation, and Entrepreneurial Systems Minitrack,"2016 49th Hawaii International Conference on System Sciences (HICSS), 2016, pp.4010-4010, doi: 10.1109/HICSS.2016.497.

[29] L. Perrin and A. Brisson, "Enhancing transport layer security with Dynamic IdentityVerification and Authentication (DIVA): Maintaining and enhancing SSL/TLS reliability," 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing,Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of

People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 2017, pp. 1-5, doi: 10.1109/UIC-ATC.2017.8397606.

[30] M. Veeningen, B. Weger, and N. Zannone. “Modeling Identity-Related Properties and Their Privacy Strength”. In: Formal Aspects of Security and Trust. Lecture Notes in Computer Science. 2011, pp. 126–140 (cited on page 56).

[31] https://www.st.com/en/evaluation-tools/32f746gdiscovery.html

[32] C. Liu, Q. Meng, T. Liao, X. Bao and C. Xu, "A Flexible Hardware Architecture for Slave Device of I2C Bus," 2019 International Conference on Electronic Engineering and Informatics (EEI), 2019, pp. 309-313, doi: 10.1109/EEI48997.2019.00074.

[33] https://github.com/Microsoft/SEAL

[34] https://archive.ics.uci.edu/ml/datasets/Room+Occupancy+Estimation

[35] https://www.st.com/en/evaluation-tools/st25dv-discovery.html

[36] T. Ishiyama, T. Suzuki and H. Yamana, "Highly Accurate CNN Inference Using Approximate Activation Functions over Homomorphic Encryption," 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 3989-3995, doi: 10.1109/BigData50022.2020.9378372.

[37] Obla, Srinath & Gong, Xinghan & Aloufi, Asma & Hu, Peizhao & Takabi, Daniel. (2020). Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks. IEEE Access. PP. 1- 1. 10.1109/ACCESS.2020.3017436.

[38] Minelli, M. (2018). Fully homomorphic encryption for machine learning. Cryptography and Security Université Paris sciences et lettres.

[39] https://en.wikipedia.org/wiki/Chinese_remainder_theorem

[40] O. Vermesan, M. Coppola, M.D.Nava, A. Capra, G. Kornaros, R. Bahr et al.,"New waves of IoT technologies research—transcending intelligence and senses at the edge to create multi experience environments", In: Internet of things—the call of the edge. Everything Intelligent Everywhere. River Publishers, p 168, 2020.

[41] G. Kornaros, "Hardware-assisted machine learning in resource-constrained IoT environments for security: review and future prospective," IEEE Access, vol. 10, no. 1, pp. 58603–58622, 2022.

[42] D. Bakoyiannis, O. Tomoutzoglou, G. Kornaros, and M. Coppola, "From hardware- software contracts to industrial iot-cloud block-chains for security", In 2021 Smart Systems Integration (SSI), pages 1–4, 2021.

[43] D. Mbakoyiannis, O. Tomoutzoglou, and G. Kornaros, "Secure over-the-air firmware updating for automotive electronic control units", In 34 th ACM/SIGAPP Symp. On Appl. Comp., SAC '19, page 174–181, 2019.

[44] G. Kornaros et al., "Towards holistic secure networking in connected vehicles through securing CAN-bus communication and firmware-over-the-air updating," ournal of System Architecture, vol. 109, Oct. 2020, Art. no. 101761.

[45] S. Leivadaros, G. Kornaros, M. Coppola, "Secure asset tracking in manufacturing through employing iota distributed ledger technology", in Procs of IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing, p. 754-761, 2021 DOI:10.1109/CCGrid51090.2021.00091

[46] G.I. Trouli and G. Kornaros, "Automotive virtual in-sensor analytics for securing vehicular communication", IEEE Design Test, vol. 37, no. 3, pp. 91–98, Jun. 2020.

[47] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek and N. Aaraj, "Survey on Fully Homomorphic Encryption, Theory, and Applications," in Proceedings of the IEEE, vol. 110, no. 10, pp. 1572-1609, Oct. 2022, doi: 10.1109/JPROC.2022.3205665.

[48] Shafagh, H., Hithnawi, A., Burkhalter, L., Fischli, P., & Duquennoy, S. (2017). Secure Sharing of Partially Homomorphic Encrypted IoT Data. Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems - SenSys ’17. doi:10.1145/3131672.3131697

[49] Baharon, M. R., Shi, Q., & Llewellyn-Jones, D. (2015). A New Lightweight Homomorphic Encryption Scheme for Mobile Cloud Computing. 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. doi:10.1109/cit/iucc/dasc/pico

[50] Boudguiga, A.; Stan, O.; Fazzat, A.; Labiod, H. and Clet, P. (2021). Privacy Preserving Services for Intelligent Transportation Systems with Homomorphi Encryption. In Proceedings of the 7th International Conference on Information Systems Security and Privacy - ICISSP, ISBN 978-989-758-491-6; ISSN 2184-4356, pages 684-693. DOI: 10.5220/0010349706840693

[51] Boudguiga, A.; Stan, O.; Fazzat, A.; Labiod, H. and Clet, P. (2021). Privacy Preserving Services for Intelligent Transportation Systems with Homomorphic Encryption. In Proceedings of the 7th International Conference on Information Systems Security and Privacy - ICISSP, ISBN 978-989-758-491-6; ISSN 2184-4356, pages 684-693. DOI: 10.5220/0010349706840693

[52] Furkan Aydin and Aydin Aysu. 2022. Exposing Side-Channel Leakage of SEAL Homomorphic Encryption Library. In Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security (ASHES'22). Association for Computing Machinery, New York, NY, USA, 95–100. https://doi.org/10.1145/3560834.3563833

[53] https://educationspaceconsultancy.com/what-are-space-frequency-occupancy-and-utilisation-rates-and-how-do-i-calculate-them/

10 Appendix

Appendix A

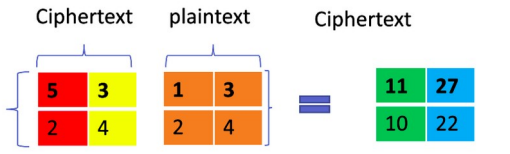
Left connectors						Right connectors				
CN No.	**Pin No.**	**Pin name**	**STM32 pin**	**Function**		**Function**	**STM32 pin**	**Pin name**	**Pin No.**	**CN No.**
						I2C1_SCL	PB8	D15	10	CN7 digital
						I2C1_SDA	PB9	D14	9	
						AVDD	-	AREF	8	
						Ground	-	GND	7	
CN6 power	1	NC	-	-		SPI2_SCK	PI1	D13	6	
	2	IOREF	-	3.3V Ref		SPI2_MISO	PB14	D12	5	
	3	RESET	NRST	RESET		TIM12_CH2, SPI2_MOSI	PB15	D11	4	
	4	+3V3	-	3.3V input/output		TIM1_CH1	PA8	D10	3	
	5	+5V	-	5V output		TIM2_CH1	PA15	D9	2	
	6	GND	-	Ground		-	PI2	D8	1	
	7	GND	-	Ground		-				
	8	VIN	-	Power input		-	PI3	D7	8	CN4 digital
-						TIM12_CH1	PH6	D6	7	
CN5 analog	1	A0	PA0	ADC3_IN0		TIM5_CH4,SPI2_NSS	PI0	D5	6	
	2	A1	PF10	ADC3_IN8		-	PG7	D4	5	
	3	A2	PF9	ADC3_IN7		TIM3_CH1	PB4	D3	4	
	4	A3	PF8	ADC3_IN6		-	PG6	D2	3	
	5	A4	PF7 or PB(1)	ADC3_IN5 (PF7) or I2C1_SDA (PB9)		USART6_TX	PC6	D1	2	
	6	A5	PF6 or PB8(1)	ADC3_IN4 (PC0) or I2C1_SCL (PB8)		USART6_RX	PC7	D0	1	

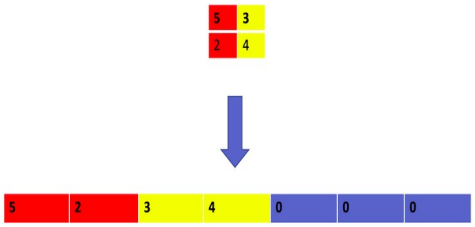
10.1 Appendix B

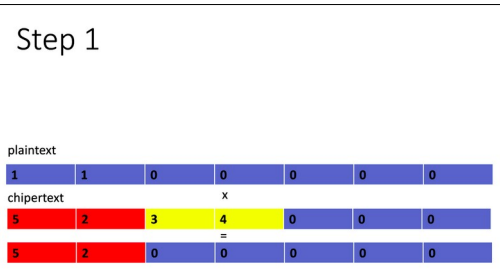
```
Ciphertext mul_vmat(Ciphertext mat, vector<vector<double>> mat1, EncryptionParameters params,
            int cols, int rows, int col2, int scale, GaloisKeys gal_keys)

{
   Ciphertext res;
   Plaintext pt;
   SEALContext context(params);
   Evaluator evaluator(context);
   CKKSEncoder encoder(context);

   vector<Ciphertext> ctA_result(cols);
   vector<Ciphertext> ctB_result(cols);

   vector<double> p(col2 * rows);

   for (int y = 0; y < col2 * rows; y++) p[y] = 0.0001;
   for (int y = 0; y < rows; y++) p[y] = 1;
   encoder.encode(p, scale, pt);

   parms_id_type last_parms_id = mat.parms_id();
   evaluator.mod_switch_to_inplace(pt, last_parms_id);

   for (int i = 0; i < cols; i++)
   {
      evaluator.multiply_plain(mat, pt, ctA_result[i]);
      evaluator.rotate_vector(mat, rows, gal_keys, mat);

      ctB_result[i] = ctA_result[i];

      for (int ii = 0; ii < cols - 1; ii++)
      {
         evaluator.rotate_vector(ctA_result[i], -rows, gal_keys, ctA_result[i]);
         evaluator.add(ctB_result[i], ctA_result[i], ctB_result[i]);
      }
   }

   vector<vector<double>> a(cols, vector<double>(rows * col2));

   for (int i = 0; i < cols; i++)
   {
      int cnt = 0;
      for (int ii = 0; ii < rows * col2;)
      {
         a[i][ii] = mat1[i][cnt];
         ii++;
         if (ii % rows == 0) cnt++;
      }
   }

   for (int i = 0; i < cols; i++)
   {
      encoder.encode(a[i], scale, pt);
      parms_id_type last_parms_id = ctB_result[i].parms_id();
      evaluator.mod_switch_to_inplace(pt, last_parms_id);
      evaluator.multiply_plain(ctB_result[i], pt, ctB_result[i]);
   }
```

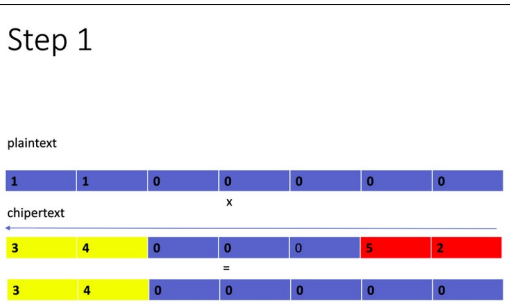
```
    evaluator.add_many(ctB_result, res);

    return res;
}
```

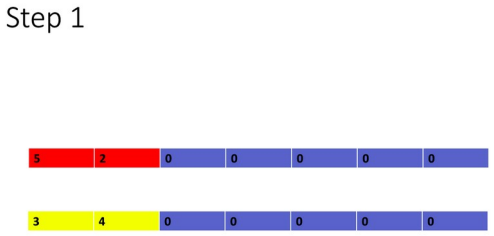
10.2 Appendix C

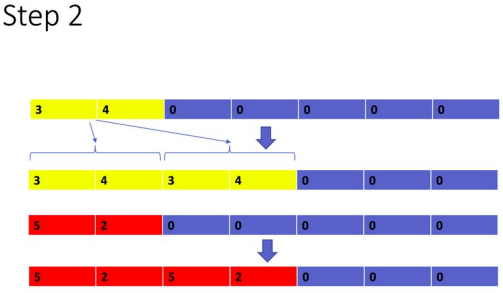


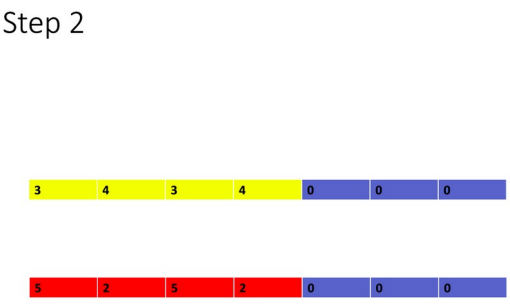


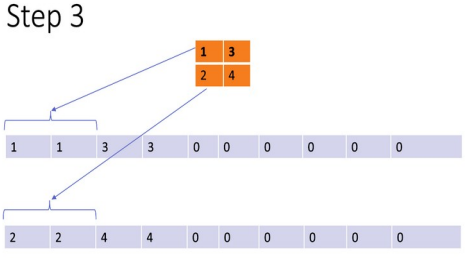


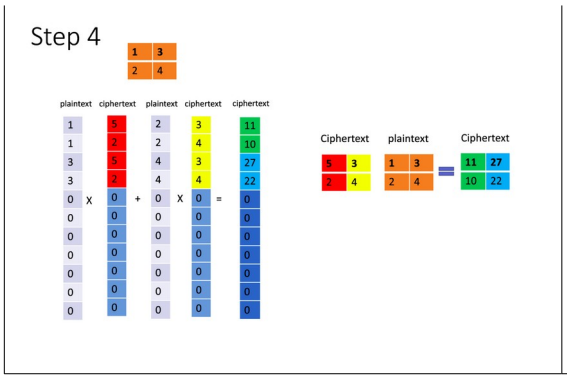












10.3 Appendix D

```
vector<vector<double>> fl_w = {
   {-0.05, 0.723, -0.044, 0.749, -0.818, 0.138, -0.076, 0.87, 0.611, 0.234, -0.59, 0.649},
   {0.17, 0.585, -0.075, 0.589, -0.202, -0.065, 0.249, 0.274, 0.353, -0.24, -0.257, 0.716},
   {-0.686, 0.799, 0.966, 0.334, -0.967, 0.612, 0.179, 0.616, -0.297, -0.429, -0.213, 0.425},
   {-0.091, -0.161, 0.13, 0.08, 0.229, 0.025, -0.975, -0.176, 1.266, 0.654, -0.05, -0.354},
   {-0.29, 1.082, 0.794, 1.03, -0.859, 0.892, 0.642, 1.189, -0.536, -0.777, -0.634, 0.715},
   {-0.611, -0.106, 0.713, 0.778, 0.005, -0.002, 0.303, 0.501, 0.06, -0.183, -0.501, 0.788},
   {-0.177, -0.01, 0.082, 0.016, -0.271, 0.064, -0.135, -0.073, 0.361, -0.417, -0.234, -0.05},
   {-0.112, 0.355, -0.086, -0.007, 0.141, -0.059, -0.203, 0.034, 0.401, -0.199, 0.176, 0.114},
   {-0.655, 0.776, -0.002, 0.226, -0.291, 0.51, 0.496, 0.281, -0.077, -0.399, -0.954, 1.308},
   {-0.376, 0.449, 0.723, 0.506, -0.319, 0.209, 0.465, 0.337, -0.602, -0.253, -1.184, 0.632},
   {-0.238, 0.297, 0.006, 0.518, -0.133, 0.454, 0.285, 0.183, 0.144, 0.066, -0.18, 0.52},
   {-0.163, 0.669, -0.079, 0.212, -0.28, -0.188, -0.675, 0.384, -0.148, 0.309, 0.302, 0.445},
   {-0.118, 0.03, 0.506, -0.085, 0.045, 0.049, -0.102, -0.215, 0.687, 0.427, -0.514, 1.04},
   {-0.928, 1.167, 0.769, 1.014, -0.851, 0.455, -0.264, 0.579, -0.305, 0.441, 0.146, -0.4},
   {-0.358, 0.281, 0.137, 0.003, -0.314, 0.484, -0.218, 0.187, 0.078, 0.025, 0.072, 0.691},
   {0.037, 0.145, 0.309, -0.442, -0.354, 0.371, 0.076, -0.199, -0.163, -0.175, -0.433, 0.514}};

vector<double> bias1 = {-0.099, 0.078, 0.168, 0.052, -0.234, 0.19,
                        -0.895, 0.25,  0.866, 1.,    0.625,  -0.844};

vector<vector<double>> sl_w = {

   {-0.479, 0.773},  {-1.074, 0.175}, {-0.878, 0.819}, {-0.828, 0.501}, {-0.87, 0.227},
   {-0.808, -0.278}, {0.717, -0.895}, {-0.825, 0.47},  {0.947, -1.366}, {1.439, -0.764},
   {0.379, -1.13},   {1.556, -1.564}

};

vector<double> bias2 = {1.036, -1.036};

Ciphertext infer(Ciphertext input, EncryptionParameters params, double scale, GaloisKeys
gal_keys,
                 SEAL::SEALContext context, RelinKeys relin_keys)
{
   Ciphertext ct;
   Plaintext pt;
   Evaluator evaluator(context);
   CKKSEncoder encoder(context);

   vector<double> tr;
```

```
    vector<double> sub;

    for (int i = 0; i < 16; i++) tr.push_back(1 / (norm_max[i % 16] - norm_min[i % 16]));

    for (int i = 0; i < 4096; i++) sub.push_back(-1 * norm_min[i % 16] + 0.0001);

    encoder.encode(sub, scale, pt);
    evaluator.add_plain(input, pt, ct);
    encoder.encode(tr, scale, pt);

    parms_id_type last_parms_id = ct.parms_id();
    evaluator.mod_switch_to_inplace(pt, last_parms_id);
    evaluator.multiply_plain(ct, pt, ct);
    evaluator.rescale_to_next_inplace(ct);

    ct = mul_vmat(ct, fl_w, params, 16, 1, 12, ct.scale(), gal_keys);

    evaluator.rescale_to_next_inplace(ct);

    encoder.encode(bias1, ct.scale(), pt);
    last_parms_id = ct.parms_id();
    evaluator.mod_switch_to_inplace(pt, last_parms_id);
    evaluator.add_plain(ct, pt, ct);

    evaluator.square_inplace(ct);

    evaluator.relinearize_inplace(ct, relin_keys);
    evaluator.rescale_to_next_inplace(ct);

    ct = mul_vmat(ct, sl_w, params, 12, 1, 2, scale, gal_keys);

    encoder.encode(bias2, ct.scale(), pt);
    last_parms_id = ct.parms_id();
    evaluator.mod_switch_to_inplace(pt, last_parms_id);
    evaluator.add_plain(ct, pt, ct);

    return ct;
}
```