# UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

**Escuela de Ciencias Matemáticas y Computacionales**

## TÍTULO:

## A Data Marketplace System for Prediction and Statistical Analysis over Sensitive Data

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información


**AUTOR:**

Serrano Palacio Nicolás Enrique

**TUTOR:**

Ph.D. Cuenca Lucero Fredy


Urcuquí, abril 2021

Urcuquí, 20 de abril de 2021

# SECRETARÍA GENERAL
## (Vicerrectorado Académico/Cancillería)
## ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
## CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
## ACTA DE DEFENSA No. UITEY-ITE-2021-00004-AD

A los 20 días del mes de abril de 2021, a las 14:30 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

| | |
|---|---|
| **Presidente Tribunal de Defensa** | Dr. ANTON CASTRO , FRANCESC , Ph.D. |
| **Miembro No Tutor** | Dr. LOPEZ RIOS, JUAN CARLOS , Ph.D. |
| **Tutor** | Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D. |

El(la) señor(ita) estudiante **SERRANO PALACIO, NICOLAS ENRIQUE**, con cédula de identidad No. **0105501878**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **A DATA MARKETPLACE SYSTEM FOR PREDICTION AND STATISTICAL ANALYSIS OVER SENSITIVE DATA** , previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

| | |
|---|---|
| **Tutor** | Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D. |

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

| Tipo | Docente | Calificación |
|---|---|---|
| Presidente Tribunal De Defensa | Dr. ANTON CASTRO , FRANCESC , Ph.D. | 9,5 |
| Tutor | Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D. | 10,0 |
| Miembro Tribunal De Defensa | Dr. LOPEZ RIOS, JUAN CARLOS , Ph.D. | 10,0 |

Lo que da un promedio de: **9.8 (Nueve punto Ocho)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

SERRANO PALACIO, NICOLAS ENRIQUE
**Estudiante**

FRANCESC ANTON CASTRO
Signé électroniquement par FRANCESC ANTON CASTRO
cn=FRANCESC ANTON CASTRO, ou= ENTIDAD DE CERTIFICACION DE INFORMACION, o= SECURITY DATA S.A. 1, c= EC
Date: 2021.04.20 15:54:31 ECT

Dr. ANTON CASTRO , FRANCESC , Ph.D.
**Presidente Tribunal de Defensa**

Firmado electrónicamente por:
**NICOLAS ENRIQUE SERRANO PALACIO**

Firmado electrónicamente por:
**FREDY ENRIQUE CUENCA LUCERO**

Dr. CUENCA LUCERO, FREDY ENRIQUE , Ph.D.
**Tutor**

Dr. LOPEZ RIOS, JUAN CARLOS , Ph.D.
**Miembro No Tutor**

DAYSY MARGARITA MEDINA BRITO
Firmado digitalmente por DAYSY MARGARITA MEDINA BRITO
Fecha: 2021.04.20 15:50:51 -05'00'

MEDINA BRITO, DAYSY MARGARITA
**Secretario Ad-hoc**

Firmado electrónicamente por:
**JUAN CARLOS LOPEZ RIOS**

# Autoría

Yo, **Nicolás Enrique Serrano Palacio**, con cédula de identidad **0105501878**, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, marzo del 2021.

Firmado electrónicamente por:
**NICOLAS ENRIQUE SERRANO PALACIO**

Nicolás Enrique Serrano Palacio
CI: 0105501878

# Autorización de publicación

Yo, **Nicolás Enrique Serrano Palacio**, con cédula de identidad **0105501878**, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, Marzo del 2021.

Firmado electrónicamente por:
NICOLAS ENRIQUE
SERRANO PALACIO

Nicolás Enrique Serrano Palacio
CI: 0105501878

# Dedication

*"To all developers and computer scientists. I hope one day this work help us to build more privacy preserving technology for the benefit of humanity and the planet."*

# Acknowledgments

Thanks to you, the readers, the ones that have follow my journey through all this time and the ones who just joined me today. To my family who have supported me in this incredible journey through science, entrepreneurship and friendship. To my professors, the ones that have showed me the true power of conviction and solidarity by giving us the opportunities to succeed in many fields and trips. To my friends with whom we have share incredible adventures in and off campus. To Majo who has been by my side supporting me in crazy situations.

# Abstract

A data marketplace is a system that enable trading among those who expect to monetize their data and those interested in gaining insights from the acquired data. In exchange for a payment, it is possible to buy advertising, demographics, public health, business intelligence and sensor data from a data owner via a data marketplace. Unfortunately, the paradigm that drives current marketplaces suffers from data leakage: one who buys data can, in principle, resell the acquired data as many times as he wants, even despising non-disclosure agreements.

Just as copyright ownership confers the author exclusive right to use his work, thus preventing others from commercializing it without the author's consent, we strongly believe that data owners should also have exclusive right on the data they struggle to collect, clean and store.

Not content with trusting in the good faith of the data buyers, we have developed a privacy-preserving data marketplace system, which allows to sell data that can be computed, though not unveiled. First, an owner provides encrypted data to a buyer, who can perform arbitrary operations on this encrypted data as if it were regular data. Thanks to homomorphic encryption, the encrypted results obtained in the buyer-side can then be decrypted in the owner-side, in a second and definitive data exchange.

The implementation uses an homomorphic encryption scheme for arithmetic of approximate numbers; making it a non deterministic solution. The final results have a small noise bounded by a constant $B$ that depends on the used encryption parameters. Therefore, this work should be used for statistical and prediction analysis that would not require exact results.

This research has built a functional data marketplace and tested it with two toy examples that allowed us to verify that data buyers can do both: calculate aggregate values and train a logistic regression-based prediction model from an encrypted data set. The capability for preventing data misappropriation might foster a paradigm shift in data trading.

***Keywords***: Data Marketplace, Private Computation, Homomorphic Encryption, Decentralized Ledgers, Machine Learning, etc.

# Resumen

Un mercado de datos es un sistema que permite una transacción entre aquellos que desean monetizar sus datos y aquellos interesados en conseguir información de los datos adquiridos. A cambio de un pago, es posible comprar datos de publicidad, demografía, salud pública, inteligencia de negocios y sensores en un mercado de datos. Lamentablemente, el paradigma dominante de los mercados de datos actuales sufre de filtración de datos: el que compra datos puede, en teoría, revender esos datos múltiples veces incluso ignorando un acuerdo de confidencialidad.

Al igual que la propiedad intelecual confiere derechos exclusivos al autor, y por lo tanto previene que la invención sea comercializada sin su consentimiento, nosotros creemos que los dueños de los datos deberían tener propiedad exclusiva sobre los datos que colectan, limpian y almacenan.

Hemos desarrollado un sistema de mercado de datos que preserva la privacidad de los mismos al permitir vender datos que pueden ser computados pero no descubiertos. Primero, un vendedor envía datos encriptados a un cliente, quien computa operaciones arbitrarias en datos encriptados como si fueran datos regulares. Gracias a la encriptación homomórfica, los resultados obtenidos del lado del comprador pueden ser desencriptados en el lado del vendedor en un segundo y definitivo intercambio de datos.

Contamos con un prototipo funcional y lo hemos probado con dos diferentes ejemplos que nos permitieron verificar el uso del mismo en: un cálculo de valores agregados y un entrenamiento de un modelo de predicción basado en regresión logística con datos encriptados. La capacidad de prevenir filtración de datos podría cambiar el paradigma de la venta de datos.

***Palabras Clave***: Mercado de Datos, Computación Privada, Encriptación Homomórfica, Libros Descentralizados, Aprendizaje de Máquina, etc.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

A couple of years ago, I was asked to visit Ortiz Corporation headquarters for a business meeting. A couple of friends and I had founded a startup called *Innomaps* that focused on providing business intelligence to small businesses and entrepreneurs through the use of data and machine learning. We gathered data from different sources and structured it in a way that was easy for the machine learning algorithms to process it. The corporation's representatives were interested in our refined data sets for their internal management strategies and were willing to pay us for it.

At *Innomaps*, we were excited to work with a corporation so we immediately decided to take the deal. Just after the meeting, when we talked to the corporation's software department about the technical details, the following question arised: **Who should share what? Should Innomaps give away its raw data to the corporation? Or, should the corporation give Innomaps the source code with which Innomaps data is to be processed?**

On the one hand, *Innomaps* data sets took time and resources to be collected and structured in an easy-to-fetch database. All this work had a cost and if we shared the data sets, we were risking that corporation (or their employees) could sell again the data sets by a simple copy-paste action, to other data buyers without Innomaps receiving a penny for it.

On the other hand, the corporation had developed its own back-end application to get the information they needed from our data set. This was their private property and if they passed it to us to compute their results, they were risking that we could use or sell the code later. There was also the risk that we could intentionally or non-intentionally run the wrong code and send them wrong results.

Thus, we reached a point of conflicting interests. None of the parties wanted to disclose and risk their confidential property but both of the parties wanted the deal to happen: the corporation needed the data sets for a proper management strategy and *Innomaps* needed to be sure that the sold data was not going to be resold without our consent.

This is the story behind the motivation of this thesis. we wanted to find a way in which both parties could protect their confidential property and allow a data sharing transaction

to happen. So that the answer of **Who should share what?** would be **No one has to share anything**.

## 1.2 Data Marketplaces

Nowadays data trading is supported by data marketplaces, which are tools that can be used to commercialize data with other stakeholders [1]. There are three main categories of data marketplaces: private, consortium-based and independent [2].

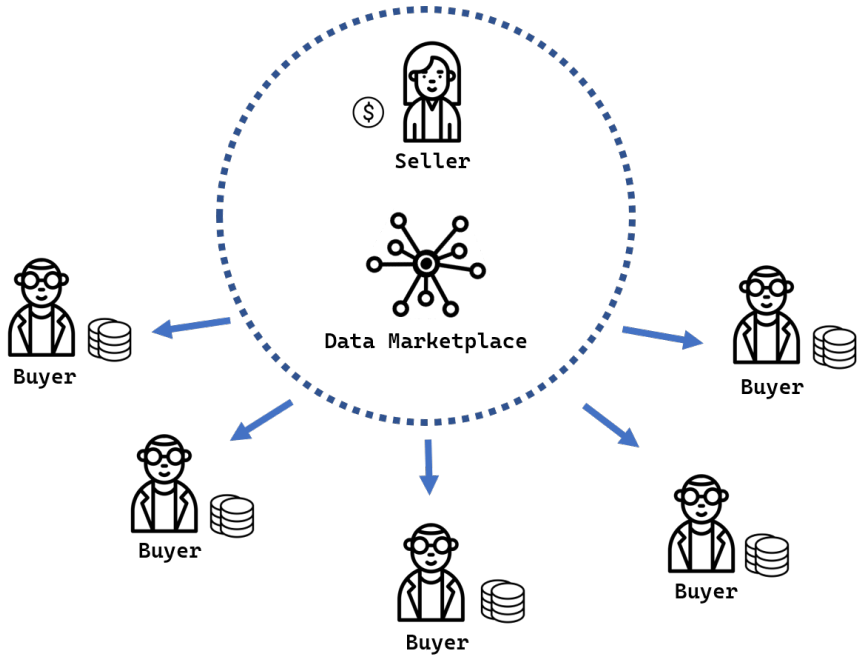


Figure 1.1: Representation of a private seller-side marketplace tool

There are different types of data marketplaces depending on who manages the software: the buyer, the seller or an intermediary[2]. Since the data marketplace proposed in this thesis is to be managed by Innomaps, it will be formally classified as a seller-side data marketplace system (see Figure 1.1).

Through this marketplace, Innomaps is able to sell data (e.g. location, lifetime, number of customers of a business, etc.) in a peer-to-peer manner.

More deeply, the buyer has to send a petition through a back-end application –a Python program using a set of APIs located and described in Innomaps webpage. Right after, Innomaps will send encrypted data to the buyer. Given that Innomaps is the only one who owns the decryption key, the buyer will never be able to decypher nor sell the data. That does not mean the data is useless. The buyer can still perform arbitrary operations with the encrypted data to obtain an encrypted result (e.g. a single encrypted value or an array of encrypted values). Once the encrypted result is decrypted, the final result is the same as if all the operations were performed on regular data. That is the idea behind homomorphic encryption.

The back-end application runs on the buyer's computer; it connects to Innomaps twice: (1) to ask for data and (2) to ask for decryption of the encrypted result. The payment and potential reimbursements are managed by a software called descentralized ledger, a neutral agent in charge of tracking the state of the buyer-seller interaction.

This work is organized as follow: Chapter 2 describes the theoretical background of homomorphic encryption which allows operations over encrypted data. Chapter 3 describes the theory behind decentralized ledgers used as an unbiased supervisor of the process. Chapter 4 describes similar related works and implementations. Chapter 5 discuses the methodology used to build the data marketplace system. Chapter 6 analyzes results from two different examples. Chapter 7 synthesize the work and discuses challenges as future works.

## 1.3 Contribution

This research will contribute to the field of data sharing by proposing a technological infrastructure that allows a trustworthy collaboration among data sellers and data buyers. We developed a black box where buyers can extract information from the data they are paying for, whereas sellers do not have to disclose their data.

### 1.3.1 Specific Objectives

1. Ensure the seller original dataset is not discoverable.
2. Ensure the buyer can compute primitive operations over the seller dataset to perform statistical analysis.
3. Ensure the two parties respect the buying agreement in the process.
4. Ensure no third party or intermediary is directly involved in the process.

### 1.3.2 Target Users

The target users for this *data marketplace system* are data scientists, business analysts, database administrators or people with medium programming skills, who would like to extract information from a private data set belonging to an external stakeholder.

It is expected that users of the proposed system have some basic knowledge of the Python programming language because they would build a back-end application to interact with the encrypted data using this popular language [3]. This programming language was selected because it has a low-learning-curve that developers can take advantage when learning how to use the proposed solution. The encryption library is also available in C++ for more experienced developers [4].

# Chapter 2

# Homomorphic Encryption

## 2.1 Background

Homomorphic encryption refers to the idea of operating over encrypted data and obtaining the result after decryption. The first ones to mention it were Ronald L. Rivest and Leonard Adleman in 1978 [5], the same ones who proposed the RSA public key cryptosystem. On their article entitled *On Data Banks and Privacy Homomorphisms* [5] they lay out some core ideas for the development of an homomorphic cryptosystem that would allow "encrypted data to be operated on without preliminary decryption of the operands". Since then, there were theoretical contributions to the field and some schemes were created (Partial or Leveled schemes) but none of them seem practical until the contribution from Craig Gentry to bootstrap the system [6] and make a Fully Homomorphic Encryption scheme practical in real life.

In the following sections the concepts of homomorphic encryption such as the mathematical definition, schemes and implementations will be discussed.

## 2.2 Mathematical Definitions

### 2.2.1 Ring

According to [7], a ring $R$ is a set together with two binary operations $\oplus$ and $\otimes$ (called addition and multiplication) satisfying the following axioms:

1. $(R, \oplus)$ is an abelian group
2. $\otimes$ is associative: $(a \otimes b) \otimes c = a \otimes (b \otimes c)$ for all $a, b, c \in R$
3. the distributive laws hold in R are:

$$\text{for all } a, b, c \in R$$

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c) \text{ and } a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

.

### 2.2.2 Homomorphism

Let $U$ be a ring with two operations $\oplus$ and $\otimes$. Let $C$ be another ring with two operations $\oplus'$ and $\otimes'$. A homomorphism between $U$ and $C$ exists if and only if there is a function $\phi$ such that:

$$\phi : U \to C$$
$$\phi(s_1 \oplus s_2) = \phi(s_1) \oplus' \phi(s_2)$$
$$\phi(s_1 \otimes s_2) = \phi(s_1) \otimes' \phi(s_2)$$
$$\text{where } s_1, s_2 \in U \text{ and } \phi(s_1), \phi(s_2) \in C \text{ [7].}$$

The operations performed in one ring are preserved in the other ring. In a homomorphic encryption scheme, the function $\phi$ could be considered the encryption function; while its inverse $\phi^{-1}$ could be considered the decryption function. It is worth noting that in order for the encryption scheme to work, $\phi^{-1}$ should exist and:

$$\phi^{-1}(\phi(s_i)) = s_i \text{ [5].}$$

Homomorphic encryption focuses on preserving the additive and multiplicative functions between *rings* of plaintexts and ciphertexts when applying the encryption and decryption operations [8]. For an illustrative example refer to Figure 2.1. The sets of the rings can be numerical (integers, complex, etc) or non-numerical (polynomials, matrices, etc).

For a toy example of how computations can be performed over encrypted data and the result will be preserved thanks to homomorphic encryption, consider the following:

$$U =< \{u_1, u_2, \dots\};\ \odot >$$
$$C =< \{c_1, c_2, \dots\};\ \odot' >$$
$$u_1, u_2 : \text{data values to operate}$$
$$\textit{encryption} : \text{encoding function } U \to C$$
$$\textit{decryption} : \text{decoding function } C \to U$$
$$\odot : \text{additive or multiplicative operation in } U$$
$$\odot' : \text{additive or multiplicative operation in } C$$

$$c_1 = \textit{encryption}(u_1)$$
$$c_2 = \textit{encryption}(u_2)$$
$$c_3 = c_1 \odot' c_2 : \text{operation executed by buyer}$$
$$\textit{decryption}(c_3) = u_1 \odot u_2.$$

The Homomorphic encryption cryptosystems can be *prime-factorization-based* (used in cryptosystems such as RSA [9], Diffie–Hellman [10], etc), *discrete-logarithm-based* (e.g.:

Elliptic-curve [11]) or *lattice-based* constructions. The last one appear to be quantum resistant as the time of writing [12]).

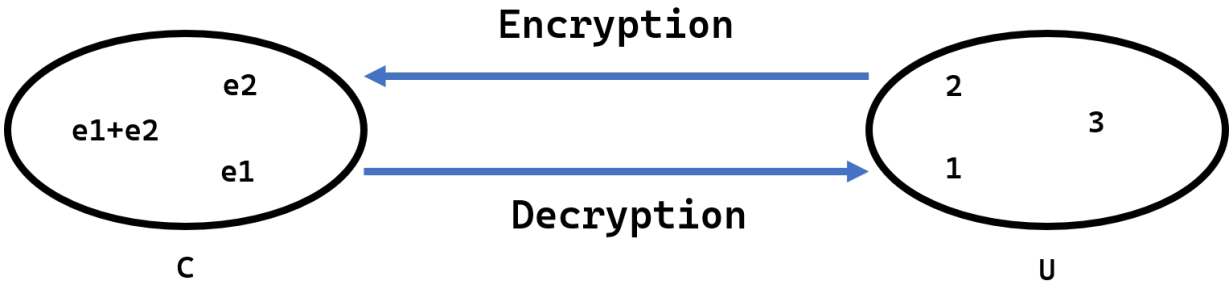


Figure 2.1: Homomorphism between a ring $C$ and a ring $U$

## 2.3 Homomorphic Encryption Schemes

A homomorphic encryption scheme defines which operations can be performed over encrypted data and how many times these operations can be applied. Particularly, any Boolean circuit can be represented using *XOR* (addition) and *AND* (multiplication) gates [13] therefore most homomorphic encryption schemes focus on defining the additive or multiplicative functions as core primitives because they are functionally complete over finite sets [13] and can be used to build more complex functions (complex circuits, function composition and even function approximation using series or polynomials).

There are three schemes: partial homomorphic encryption, somewhat homomorphic encryption and fully homomorphic encryption.

### 2.3.1 Partial Homomorphic Encryption

This scheme refers to cryptosystems that can operate just one operation over encrypted data in an unlimited number of times. In the [13] survey, Acar et al. describe in more detail multiple Partial Homomorphic Encryption examples that either have the additive function or the multiplicative function defined.

A specific and interesting example of this type of scheme is the Paillier cryptosystem [14]. Pascal Paillier, the author, introduces an innovative probabilistic encryption scheme based on the *composite residuosity problem* and build an homomorphic encryption cryptosystem with interesting additional properties. The system not only can perform additive operations between ciphertext-ciphertext and plaintext-ciphertext, but also multiplicative operations between plaintexts and ciphertexts. This make it practical for real use cases [13].

### 2.3.2 Somewhat Homomorphic Encryption

The Somewhat Homomorphic Encryption schemes specify cryptosystems that allow some types of operations a limited number of times [13]. It is worth nothing that since the

appearance of homomorphic encryption in 1978 [5], researchers and contributors have been trying to achieve a Fully Homomorphic Encryption; therefore the Somewhat Homomorphic Encryption schemes are just an attempt to allow multiple operations in unlimited number of times. The problem with the examples under this scheme is the ciphertext growth when operating specific functions (in most cases present in [13] multiplication) making the result corrupted at the moment of decryption.

The ciphertext growth occurs because every pair of encrypted messages already have some level of noise to protect them against attacks (e.g.: semantic attacks) and when multiplying these messages together, the noise is also multiplied making the result more difficult to clean from its noise.

### 2.3.3 Fully Homomorphic Encryption

In 2009, Craig Gentry [6] found a way to solve the ciphertext growth problem. He proposed a bootstrapping technique that would allow the cryptosystem to decrypt the messages, perform the operations and encrypt it again without leaking any information in the process. This bootstrapping technique was possible thanks to a squashing method that allow any arbitrary-sized ciphertext to be reduced to a depth level that the bootstrapping technique could handle. After squashing, the bootstrapping involved a *recrypting* procedure of the ciphertext by the use of another set of keys provided by the encrypting stakeholder [13].

For a visual example imagine a jewelry factory owner that wants its employers to transform gold pieces into jewelry. She is afraid the employers would steal her gold pieces so she decides to give them gloveboxes with lock to manufacture the jewelry inside. The problem she faces now is that the gloveboxes start to fill and her employees are not able to keep working. To solve it, she decides to build a complex pipeline of gloveboxes better described by Gentry himself [15]: "She gives a worker a glovebox, box #1, containing the raw materials. But she also gives him several additional gloveboxes, where box #2 contains (locked inside) the key to box #1, box #3 contains the key to box #2, and so on. To assemble an intricate design, the worker manipulates the materials in box #1 until the gloves stiffen. Then, he places box #1 inside box #2, where the latter box already contains the key to box #1. Using the gloves for box #2, he opens box #1 with the key, extracts the partially assembled trinket, and continues the assembly within box #2 until its gloves stiffen. He then places box #2 inside box #3, and so on. When the worker finally finishes his assembly inside box #n, he hands the box to the factory owner".

The problem with Gentry's bootstrapping solution is that is not always feasible in real world scenarios because of its computational complexity. Therefore, the Fully Homomorphic Encryption schemes have been subdivided in two:

#### Leveled Homomorphic Encryption

These cryptosystems try to imitate a Fully Homomorphic Encryption by setting parameters high enough to avoid a considerable ciphertext growth and therefore allow its users to compute safely multiple operations in limited number of times.

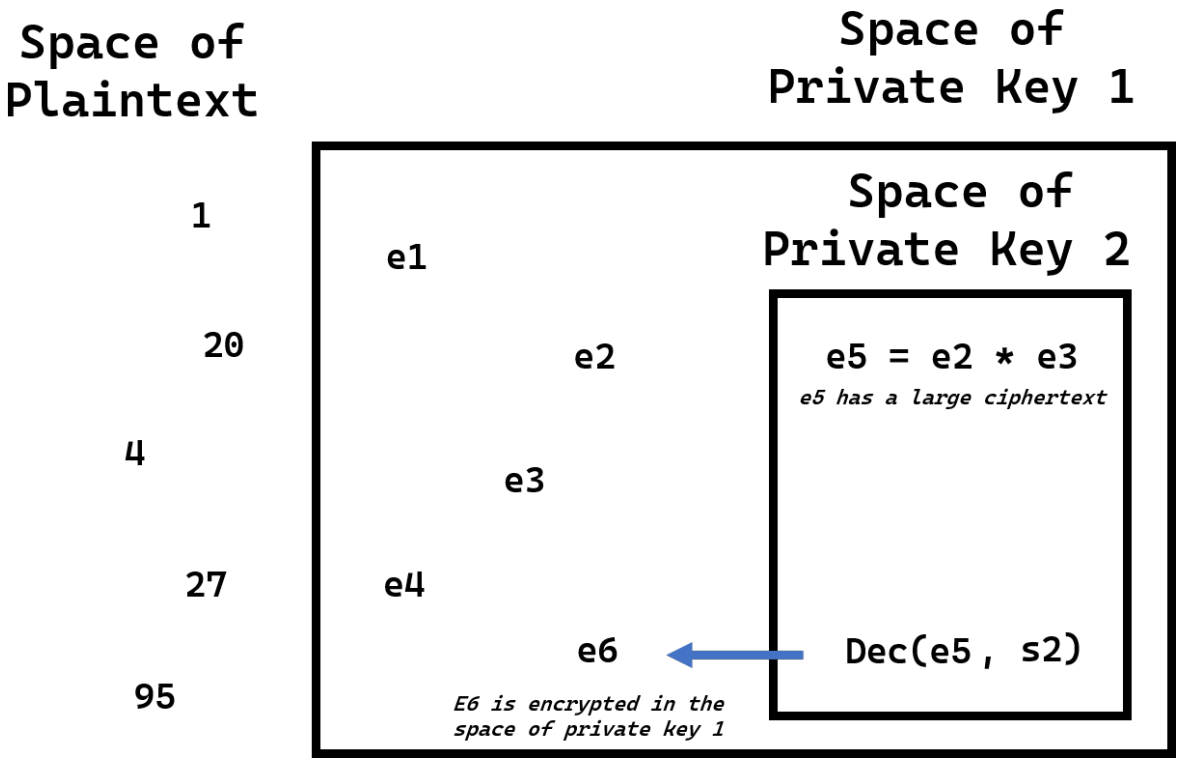


Figure 2.2: Bootstrapping's explanation where a secret key is used to decrypt a ciphertext and squash its size while keeping it encrypted from the end user

#### Truly Fully Homomorphic Encryption

These cryptosystems have tried and succeeded in implementing Gentry's bootstrapping technique or a similar solution. Therefore, these cryptosystem are considered Fully Homomorphic Encryption schemes because it allows to compute multiple operations in unlimited number of times.

## 2.4 CKKS Encryption

In this work, the Homomorphic Encryption for Arithmetic of Approximate Numbers implementation proposed by Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song [4] will be used. This implementation is called HEEAN because of the article's title initials and also called CKKS because of the authors initials. They implemented both a Leveled Fully Homomorphic Encryption scheme and a Fully Homomorphic Encryption scheme. The implementation used in this work is a Python wrapper library created by the OpenMined organization [16] for the C++ Microsoft Simple Encrypted Arithmetic Library (SEAL) which contains the CKKS [4] scheme.

As the name states, the encryption works with approximate numbers and not their real value. For this reason, the use of the CKKS encryption is limited to statistical or prediction application which does not require complete precision. It is worth mentioning that CKKS is the only fully homomorphic encryption that works over real numbers and

not only integers.

In overview, CKKS has the following phases:

1. Generating the public and private key for the encryption process
2. Grouping data as a vector $a = (a_1, a_2, a_3, ...)$ or as a set of vectors (e.g.: matrix)
3. Encoding the vector as a polynomial $p(x) = c_0 + c_1x^1 + c_2x^2 + \ldots + c_nx^n$ which is directly related to the vector of the previous step. More information about this relationship is provided at section 2.4.2.
4. Encrypting the polynomial coefficients to obtain the ciphertext
5. Performing the desired operations on the ciphertext
6. Decrypting the polynomial coefficients (the ciphertext)
7. Decoding the polynomial as a vector

CKKS uses polynomials instead of vectors because they provide an efficient and more secure way to compute operations. It also makes it easy to interact with polynomials representing function approximations. [17] [4]

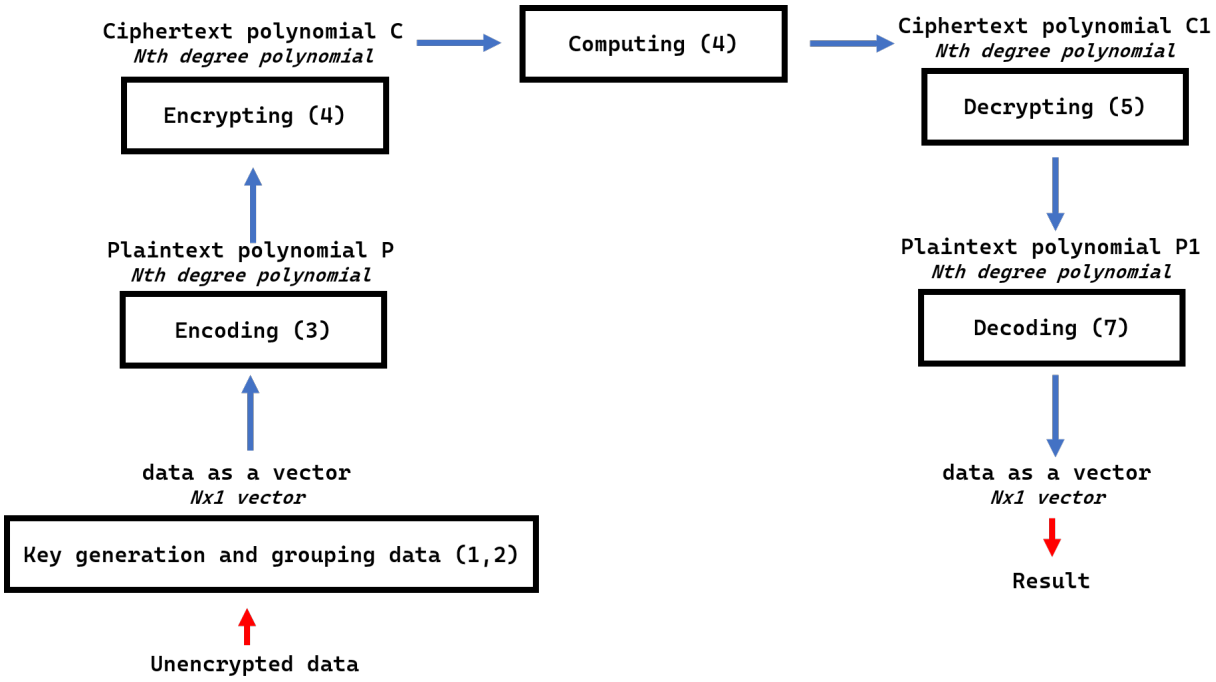


Figure 2.3: CKKS workflow

The CKKS encryption used in this work is an asymmetric encryption scheme; this means that the data owner is able to encrypt data using a public key and only decrypt it using a private key. The owner would share the public key with others but keep the private key secret. Anyone with access to the private key would be able to decrypt the data [4].

Next, we continue with detailed explanations of each of the encryption and decryption phases of CKKS.

### 2.4.1 Key generation algorithm

CKKS is based on the *Ring Learning-With-Error* problem [18]. To understand this problem, consider the simpler *Learning-With-Error* version. A vector $m$ of size $n$ can be encrypted by adding the expression $As+e$ which represents a *matrix-vector* multiplication between matrix $A$ and vector $s$ and an addition of vector $e$. So the encrypted vector would be $b = m + (As + e)$. The origin of these elements is as follows:

1. $A$: is a matrix whose elements have been uniformly sampled from $\mathbb{Z}_q$ (which represent the set of integers modulo $q$. $q$ can be a very large element).

2. $s$: is our secret $n$-sized vector which can be select as a random vector or a vector specified by the user.

3. $e$: is a $n$-sized vector containing small errors (usually guassian noises). These errors would be added to the plaintext vector and are supposed to be small enough to not represent a major difference between the decrypted ciphertext and the plaintext ($\tilde{u} = u$).

The security of the problem depends on the difficulty to find $As + e$ (or also $-As + e$) having $A$ but not having $s$ (secret vector) nor $e$ (small noises vector).

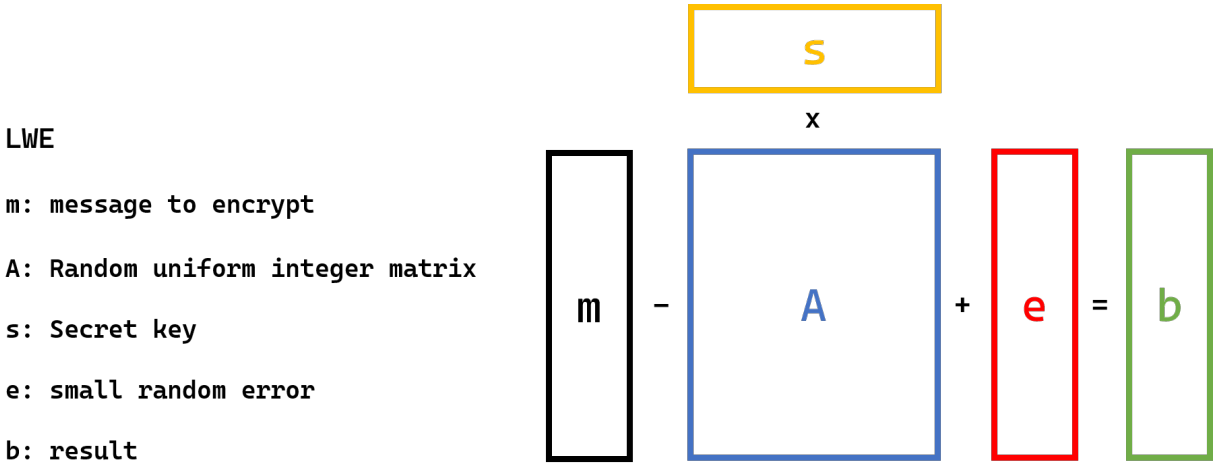


Figure 2.4: Learning-With-Errors problem

The *Ring Learning-With-Error* problem is exactly the same as the *Learning-With-Error* problem but it uses an algebraic ring. In the CKKS scheme, this algebraic ring is the polynomial ring $\mathbb{Z}_q[X]/(X^N + 1)$ (the set of all polynomials with *integers-modulo-q* coefficients, modulo the cyclotomic polynomial $X^N + 1$). It is worth mentioning that *integers-modulo-q* is defined as $\mathbb{Z}_q = \{0, 1, 2, ..., q-1\}$. So all $A$, $s$ and $e$ can be interpreted as polynomials rather than vectors and matrices. This change allows for more efficient not quadratic but linear key size and also for a multiplication complexity reduction from $\mathcal{O}(n^2)$ to $\mathcal{O}(nlog(n))$ because it can be done using more efficient methods than the *matrix-vector* multiplication [4].

To conclude the private key is the secret polynomial $s$ and the public key is the tuple $(-As + e, A)$.

### 2.4.2 Encoding and decoding algorithm

As previously mentioned, a polynomial would be needed to take advantages of the *Ring Learning-With-Error* problem so an encoding algorithm to transform a vector to a polynomial is needed.

It is easier to understand the decoding process first rather than the encoding one. To transform a polynomial to a set of numbers (vector), we can evaluate the polynomial using each of the $N$ unity roots of the cyclotomic polynomial $X^N + 1$. A possible more efficient solution would be to calculate the eigenvalues of the quotient algebra of the ring of polynomials by the ideal generated by the cyclotomic polynomial. This alternative approach is part of future works and therefore not considered to be implemented in this proposed system.

So a polynomial $p(x)$ would be decoded to the vector:

$$v = (p(\xi^1), p(\xi^3), \ldots, p(\xi^{2N-1})) = (v_1, v_2, \ldots, v_N)$$

with $\xi^{2n-1}$ as the n-th unity root of the cyclotomic polynomial.

More informally, a function $\sigma$ can be defined as follows:

$$\sigma : \mathbb{C}_q[X]/(X^N + 1) \to \mathbb{Z}_q^N$$
$$\sigma(p(x)) = (p(\xi^1), p(\xi^3), \ldots, p(\xi^{2N-1}))$$

Consequently, to encode a vector to a polynomial, the inverse $\sigma$ function is needed. A linear equation would better represent this situation with:

$V$ : the Vandermonde matrix of $\xi^{2i-1}$ with $i = 1, 2, \ldots, N$

$c$ : the polynomial coefficients vector $(c_1, c_2, c_3, ...)$

$a$ : the vector to encode or decode $(a_1, a_2, a_3, ...)$

$Vc = a$ would help to decode the polynomial to a vector

$V^{-1}a = c$ would help to encode the vector to a polynomial

It should be recalled that the domain of $\sigma$ is $\mathbb{C}_q[X]/(X^N+1)$ and the *integers-modulo-q* coefficient polynomials are needed to take advantages of the integers polynomials ring. In order to transform this domain, the CKKS scheme uses a *coordinate-wise random rounding* process which was proposed in the *Toolkit for Ring-LWE Cryptography* [19]. To avoid numbers' losses in the rounding process, the vector elements are multiplied by a constant $\Delta$ that depending on its value, it gives a $\frac{1}{\Delta}$ precision.

### 2.4.3 Encryption and decryption algorithm

After having encoded our vector ($\mu$) into an *integers-modulo-q* coefficient polynomial and having created our public ($p$) and private ($s$) keys, the encryption and decryption algorithms of our plain encoded vector are:

$$p : \text{Public key tuple } (-As + e, A)$$

$$s : \text{Private key polynomial}$$

$$\text{Encryption } enc[\mu] : (\mu, 0) + p = (\mu - As + e, A) = (c_1, c_0)$$

$$\text{Decryption } dec[(c_1, c_0)] : c_1 + c_0 s = \mu - As + e + As = \mu + e \approx \hat{\mu}$$

$$\mu + e \approx \hat{\mu}$$

The error ($e$) makes the encryption scheme an approximate arithmetic scheme while also making it hard to break. This error is sampled from a Gaussian distribution of standard deviation $\sigma = 3.2$ [4]. It is worth pointing out that the error is needed for the LWE hard assumption problem used to built the CKKS cryptosystem [4] [18]. The error is considered small enough to state that the decrypted result is an approximated value to the true result [4]. Therefore this work should be used only for statistical and prediction analysis.

Certain operations over encrypted data could make the error grow and therefore make the approximation of $\hat{\mu}$ to $\mu$ obsolete. Leveled Homomorphic Encryption (choosing encryption parameters high enough to not worry avoid error grow) and Fully Homomorphic Encryption (using a bootstrapping technique which decrypts, eliminates the noise (error) and encrypts again the ciphertext without letting the user see the decrypted data) are two current solutions to this problem [13] [6].

The TenSEAL library (a python wrapper of Microsoft implementation) used in this work does not have a bootstrapping technique implemented for CKKS yet. This issue is already part of the roadmap as a future implementation but on the time of writing, it has not been done [20]. Therefore, this work is using a Leveled Homomorphic Encryption with a possible opportunity to become a Fully Homomorphic Encryption thanks to the future bootstrapping implementation [20].

### 2.4.4 CKKS recap

To recap, the CKKS encryption and decryption method go as following:

$$\text{Let } a = (a_1, a_2, ..., a_n) \in \mathbb{R}^n \text{ be a unencrypted element}$$
$$\text{Let } b = b_0 + b_1 x + b_2 x^2 + ... + b_{n-1} x^{n-1} \text{ be an encrypted element}$$
$$\text{Let } c \in \mathbb{C}_q[X]/(X^N + 1) \text{ be the second component of the public key}$$

$$\text{Let } \sigma : \mathbb{C}_q[X]/(X^N + 1) \to \mathbb{R}^N \text{ be the decoding function described in 2.4.2}$$
$$\text{Let } \sigma^{-1} : \mathbb{R}^N \to \mathbb{C}_q[X]/(X^N + 1) \text{ be the encoding function described in 2.4.2}$$
$$\text{Let } enc : \mathbb{C}_q[X]/(X^N + 1) \to [\mathbb{C}_q[X]/(X^N + 1)]^2 \text{ be the encryption function described in 2.4.3}$$
$$\text{Let } dec : [\mathbb{C}_q[X]/(X^N + 1)]^2 \to \mathbb{C}_q[X]/(X^N + 1) \text{ be the decryption function described in 2.4.3}$$

$$ckks\_encryption : \mathbb{R}^N \to [\mathbb{C}_q[X]/(X^N + 1)]^2$$
$$ckks\_encryption(a) = enc(\sigma^{-1}(a))$$
$$ckks\_decryption : [\mathbb{C}_q[X]/(X^N + 1)]^2 \to \mathbb{R}^N$$
$$ckks\_decryption(b, c) = \sigma(dec(b, c))$$

From now on in this work, the term *enc* will refer to *ckks_encryption* and the term *dec* will refer to *ckks_decryption*. It is worth pointing out that the term *plaintext* refers to an unencrypted element and the term *ciphertext* refers to an encrypted element. The element could be a number, a vector or a polynomial. In the following sections of this work, plaintext elements would refer to vectors unless otherwise specified, and ciphertext elements would refer to polynomials unless otherwise specified.

### 2.4.5 Operations on ciphertexts

The following algorithms are used to operate over ciphertexts:

1. **Plain-Cipher Addition:**

$$\text{Let } a, b \in \mathbb{R}^n \text{ be plaintext vectors}$$
$$enc(a) = (c_1, c_0)$$
$$enc(a) + b := (c_1 + b, c_0)$$

$$\begin{aligned} dec(c_1 + b, c_0) &= c_1 + b + c_0 s \\ &= c_1 + c_0 s + b \\ &\approx a + b \end{aligned}$$

2. **Cipher-Cipher Addition:**

$$\begin{aligned} \text{Let } a, b \in \mathbb{R}^n \text{ be plaintext vectors} \\ enc(a) = (c_1, c_0) \text{ and } enc(b) = (c_1', c_0') \\ enc(a) + enc(b) := (c_1 + c_1', c_0 + c_0') \end{aligned}$$

$$\begin{aligned} dec(c_1 + c_1', c_0 + c_0') &= (c_1 + c_1') + (c_0 + c_0')s \\ &= c_1 + c_1' + c_0 s + c_0' s \\ &= c_1 + c_0 s + c_1' + c_0' s \\ &= (c_1 + c_0 s) + (c_1' + c_0' s) \\ &\approx a + b \end{aligned}$$

3. **Plain-Cipher Multiplication:**

$$\begin{aligned} \text{Let } a, b \in \mathbb{R}^n \text{ be plaintext vectors} \\ enc(a) = (c_1, c_0) \\ enc(a) * b := (bc_1, bc_0) \end{aligned}$$

$$\begin{aligned} dec(bc_1, bc_0) &= bc_1 + bc_0 s \\ &= b(c_1 + c_0 s) \\ &\approx b * a \end{aligned}$$

4. **Cipher-Cipher Multiplication:**

Let the encrypter be the one who encrypts the elements

Let $a, b \in \mathbb{R}^n$ be plaintext vectors

$enc(a) = (c_1, c_0)$ and $enc(b) = (c_1', c_0')$

$$enc(a)enc(b) := (d_0, d_1) + \frac{d_2}{p} evk$$

where:

$d_0 = c_1 c_1'$

$d_1 = c_0 c_1' + c_1 c_0'$

$d_2 = c_0 c_0'$

$evk = (-a_0 s + e_0 + ps^2, a_0)$ is a tuple given by the encrypter

$a_0$ is an uniformly sampled polynomial given

$e_0$ is a small uniformly sampled error polynomial

$p$ is a big integer

$$\begin{aligned}
dec((d_0, d_1) + \frac{d_2}{p} evk) &= dec((c_1 c_1', c_0 c_1' + c_1 c_0') + \frac{c_0 c_0'}{p}(-a_0 s + e_0 + ps^2, a_0)) \\
&= dec((c_1 c_1', c_0 c_1' + c_1 c_0') + (-\frac{c_0 c_0'}{p} a_0 s + \frac{c_0 c_0'}{p} e_0 + \frac{c_0 c_0'}{p} ps^2, \frac{c_0 c_0'}{p} a_0)) \\
&= dec(c_1 c_1' - \frac{c_0 c_0'}{p} a_0 s + \frac{c_0 c_0'}{p} e_0 + \frac{c_0 c_0'}{p} ps^2, c_0 c_1' + c_1 c_0' + \frac{c_0 c_0'}{p} a_0) \\
&= c_1 c_1' - \frac{c_0 c_0'}{p} a_0 s + \frac{c_0 c_0'}{p} e_0 + \frac{c_0 c_0'}{p} ps^2 + (c_0 c_1' + c_1 c_0' + \frac{c_0 c_0'}{p} a_0)s \\
&= c_1 c_1' - \frac{c_0 c_0'}{p} a_0 s + \frac{c_0 c_0'}{p} e_0 + \frac{c_0 c_0'}{p} ps^2 + c_0 c_1' s + c_1 c_0' s + \frac{c_0 c_0'}{p} a_0 s \\
&= c_1 c_1' + \frac{c_0 c_0'}{p} e_0 + c_0 c_0' s^2 + c_0 c_1' s + c_1 c_0' s \\
&= c_0 c_0' s^2 + (c_0 c_1' + c_1 c_0')s + c_1 c_1' + (\frac{c_0 c_0'}{p} e_0) \\
&= (c_1 + c_0 s)(c_1' + c_0' s) + (\frac{c_0 c_0'}{p} e_0) \\
&= dec(enc(a), s)\ dec(enc(b), s) + (\frac{c_0 c_0'}{p} e_0) \\
&= ab + (\frac{c_0 c_0'}{p} e_0) \\
&\approx ab \text{ when } p \text{ is a big integer and } e_0 \text{ is a small error}
\end{aligned}$$

CKKS is a Leveled Fully Homomorphic Encryption scheme so an unlimited types of operations should be allowed. The addition and multiplication operations already let the user have a subtraction (negative addition) and a division (multiplication by its respective decimal value $\frac{1}{2} = 0.5$) operations. These 4 primitive operations allow for more com-

plex functions using function approximations (logarithm, exponent, sigmoid, evaluation functions over a specific domain and more) [13].

## 2.5 Existing Implementations

The following section describes existing implementations related to homomorphic encryption and other privacy preserving systems.

### 2.5.1 Microsoft SEAL

Microsoft is developing an open source C++ library called *Simple Encrypted Arithmetic Library* or *SEAL*. According to its 2017 manual, "SEAL was created for the specific goal of providing a well-engineered and documented homomorphic encryption library, with no external dependencies, that would be easy to use both by experts and by non-experts with little or no cryptographic background" [20]. The library implements two homomorphic encryption schemes:

- BFV (Brakerski, Gentry, and Vaikuntanathan) scheme that provides modular arithmetic to be performed on encrypted integers [21].
- CKKS ( Cheon-Kim-Kim-Song) scheme that allows additions and multiplications on encrypted real or complex numbers, but yields only approximate results [21].

The library successfully accomplish its goal in the C++ programming language field but thanks to the widespread use of Python [3], the Open Mined community [16] decided to create a python wrapper [22] (called TenSEAL) for most of the SEAL library functions making both the C++ memory efficiency and the Python simplicity available together for users of the wrapper.

### 2.5.2 NuCypher

NuCypher is a crypto-startup that provides multiple privacy services focused on decentralized applications [23]. One of its services is a python library for Fully Homomorphic Encryption (including a bootstrapping implementation) called NuFHE which uses CUDA and OpenCL [24].

They have been focusing on building a Key Management System for symmetric and asymmetric encryption used in applications. Therefore, they give as much flexibility as possible to the developers by making the Fully Homomorphic Encryption implementation work with bytes directly.

### 2.5.3 CrypTen

The Facebook AI Research division open-sourced the CrypTen library in October 2019. According to its press release, this library is an easy-to-use framework built on PyTorch to facilitate research in secure and privacy-preserving machine learning [25].

This implementation primarily uses *Secure Multi Party Computation (SMPC)* as its encryption scheme [26]. As the name itself states, this encryption scheme is useful for multiple parties or stakeholders involved in the data processing phase. Applications that uses multiple data sources and need them to be private/encrypted can take advantage of the functionalities of SMPC.

### 2.5.4 HElib

HElib is an open-source C++ library that implements the BGV scheme with bootstrapping and the Approximate Number scheme of CKKS, along with many optimizations to make homomorphic evaluation run faster, focusing mostly on effective use of the Smart-Vercauteren ciphertext packing techniques and the Gentry-Halevi-Smart optimizations [27].

This thesis work uses the TenSEAL python wrapper to perform all its homomorphic encryption operations. The decision to use this implementation over the ones described above is because of its immense and helpful community, extensive tutorials and documentation, *state-of-the-art* implementations and active development.

## 2.6 Current Trends

In the following section, the current homomorphic encryption trends are described. The term *trends* refers to the multiple pathways or branches the privacy preserving research (which includes homomorphic encryption) is heading. It was important to consider these trends in the implementation phase of this thesis work in order to choose the one that fit the goal.

As Rivest stated in the first mention of homomorphic encryption [5], this type of encryption would not allow comparison operations such as smaller than, larger than, equal to. The premise was that if the user is able to perform any of this operations, he or she would have more probabilities to guess the real value. This is one of the main problems of homomorphic encryption and most of the trends described below were proposed to solve this challenge.

### 2.6.1 Circuits Approach

With the comparison problem in mind, this trend has been focused on redefining specific algorithms that use comparison operations in order to make them run in an homomorphic encryption setting. For example, the following standard algorithm:

```
1 if number == 5:
2     result = number + 3
3 else:
4     result = number + 2
```

can be translated into an homomorphic encryption circuit like:

```
comparison = (number -5)
# This variable would be encrypted

result = (1-comparison)(number + 3) + (comparison)(number + 2)
# The result would depend on the value of the comparison variable
```

This circuit approach is interesting and can be used with this thesis work but it involves a deep knowledge of the algorithm that is going to be executed and can take some time to translate from standard code into homomorphic encryption circuits.

### 2.6.2 Function Approximation Approach

This approach mainly involves the comparison operations again. The idea behind it is to build function approximations that could give the user the desired result of a comparison operation. According to the Stone–Weierstrass theorem, all mathematical continuous functions on specific interval $[a, b]$ can be approximated by one or more polynomials to any degree of precision. [28].

This approach can be easily used in this thesis work implementation because of the polynomial features of the CKKS wrapper. The user needs to know the polynomial approximation of the desired function and also the interval (minimum and maximum values) that contains the data to be analyzed.

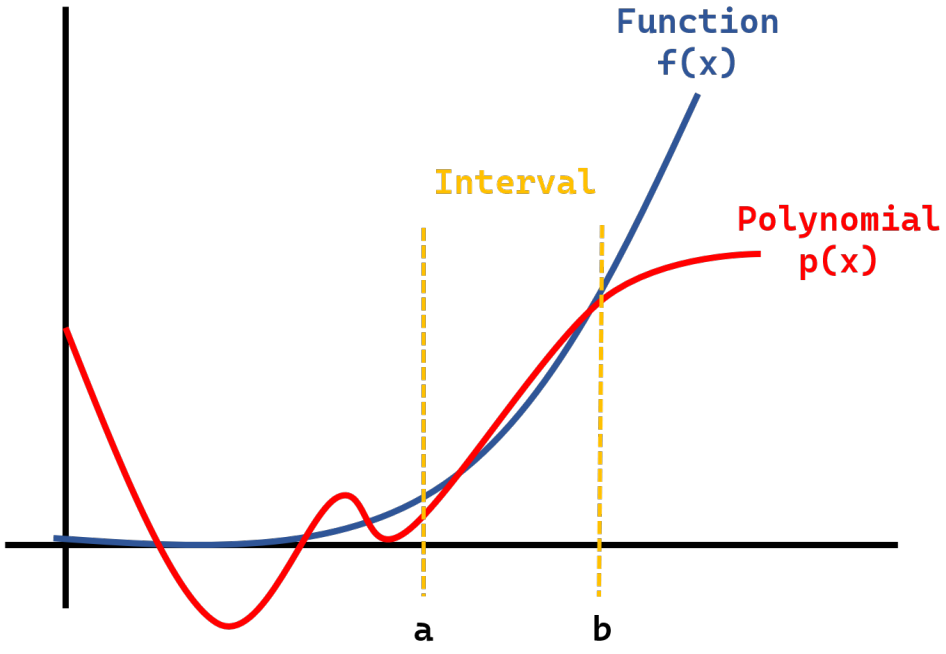


Figure 2.5: Stone-Weierstrass theorem example

### 2.6.3 Client-Server Communication Approach

To solve the comparison operations problem of homomorphic encryption, some implementations of the academic field propose a Client-Server communication approach. The process is as follows:

1. Server sends encrypted data to client
2. Client orders the system to perform a comparison operation
3. Client system computes the difference between the encrypted data and the comparison criteria
4. Client system sends the difference result to the server
5. Server system checks if values are greater than, smaller than or equal to zero and creates a boolean vector with Trues (1) and Falses (0)
6. Server system sends the boolean vector to the client
7. Client can see the results of the comparison operation

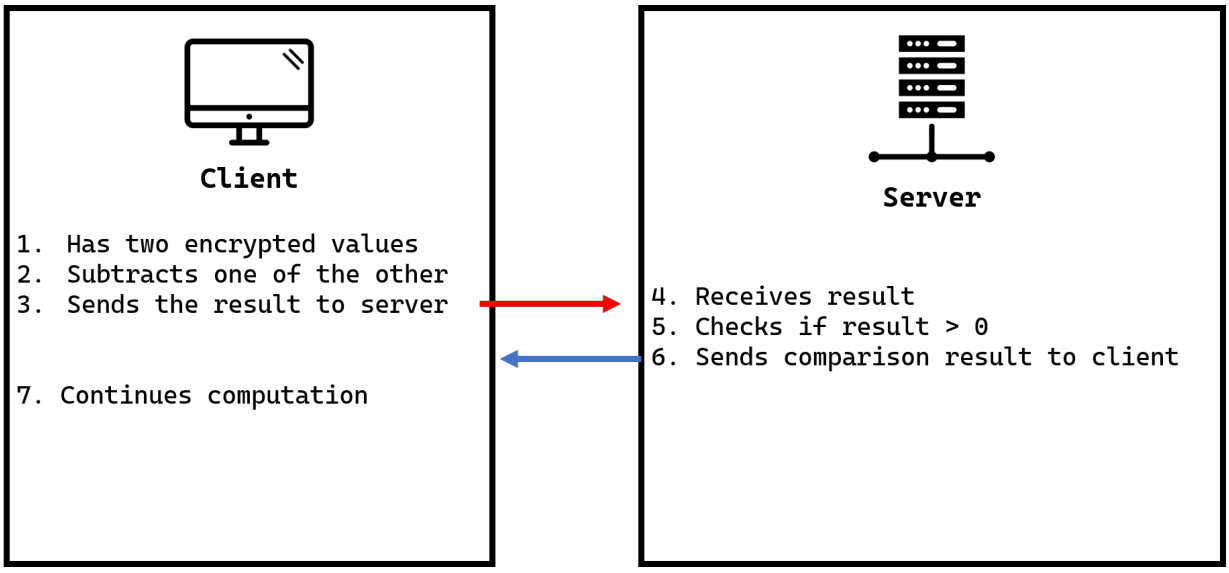


Figure 2.6: A client-server approach workflow

The issue with this approach is that it ignores Rivest's caveat of comparing encrypted values [5] and therefore the system becomes vulnerable for data leakage. Another important issue is the overhead time spend in the client server communication protocol. This issue could be solved in the future by trusting technology improvements on speed and efficiency of communication protocols [29].

### 2.6.4 Enclaves Approach

The enclaves approach is based on using secure environment extensions from the CPU architecture to compute in a confidential manner where all other running software is considered malicious (kernel, hypervisor, etc) [30].

The most common enclaves nowadays are the Intel Software Guard Extensions (SGX) which are implemented in the CPU architecture. This hardware establishes a secure container, and the user uploads the desired computation and data into the secure container. The trusted hardware protects the data's confidentiality and integrity while the computation is being performed on it [30].

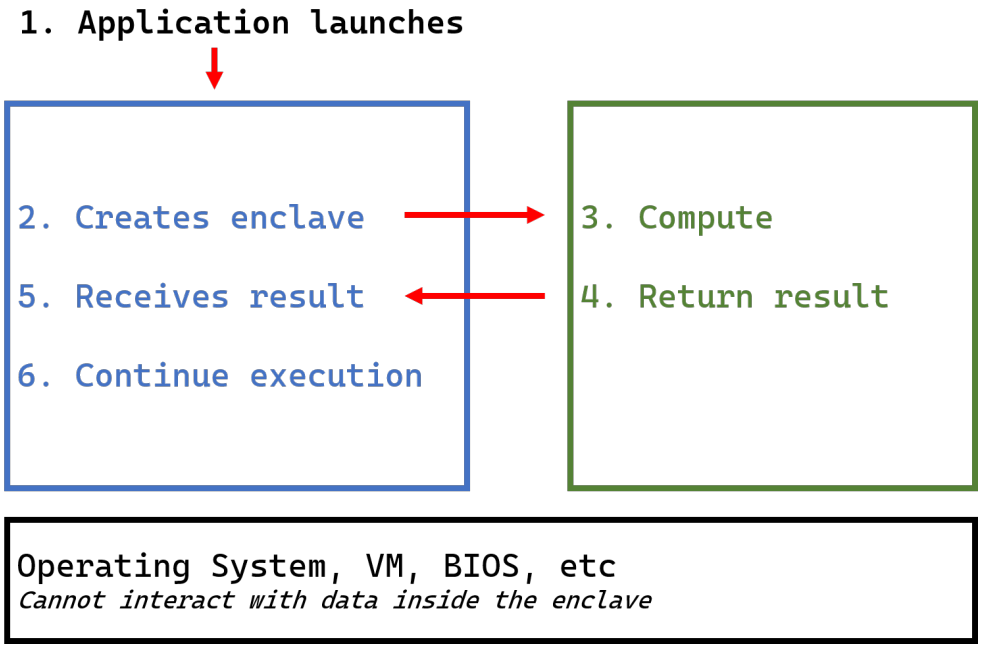


Figure 2.7: A enclave workflow

Another similar/related solution is Oblivious RAM which implements a privacy-preserving RAM memory fetching method by compiling an algorithm into a new one that preserves the input-output behaviour but the memory access interactions are different from the original algorithm. [31]

This approach allows for a private data computation without the processing overhead of homomorphic encryption. It creates a black box that the user can use to insert data to compute and the machine would not be able to see it. The problem it presents is that one of the stakeholders involved in the data transaction needs to have an Intel SGX powered hardware, Oblivious RAM compiler or similar in order to create the secure container. This limitation is considerable in multiple scenarios where the data seller does not have the enough resources to invest in the appropriate hardware or to rent resources with the required enclaves from a cloud provider.

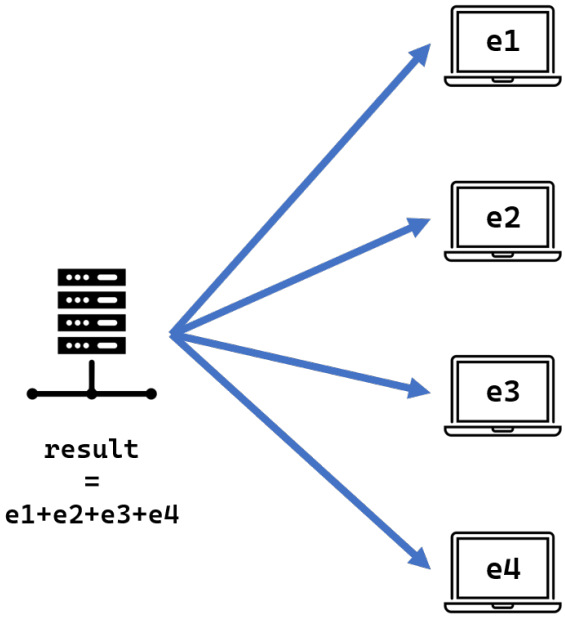


Figure 2.8: Multi-party computation between multiple independent clients

### 2.6.5 Multi Party Computation Approach

The *privacy-preserving* machine learning trend has been heavily based on homomorphic encryption and multi party computation. The later is powered by *encrypting and splitting* the data into different equal sets and send it to different parties (stakeholders or users). The encryption scheme allows for operations to be executed independently in different parties computers and later on join it together to recover the final result [16].

This approach can be used in machine learning scenarios where the data can be or is segmented in different clients nodes. The issue with this approach is that more than two parties (or stakeholders) are needed in order to successfully implement it, therefore the multi party computation does not accomplish this thesis goal.

# Chapter 3

# Decentralized Ledger

## 3.1 Background

### 3.1.1 Byzantine Generals Problem

Imagine yourself as a byzantine general in the middle of the night getting ready to attack an enemy city at the dawn of tomorrow. You are on one of the four hills surrounding the city and your other general comrades are in the other three left. A couple of days before arriving to the city, you were together in a strategic meeting where you all realized that the attack could be successful if, and only if, the four of you attacked at the same time, otherwise you would fail and your respective legions would be killed in action.

Right now, at midnight you decided to confirm that the other generals would authorize the attack at dawn and you send a messenger to each one of the hills. Each messenger has the risk of being intercepted by the enemy or being lost in the route. Even if a messenger arrives to the other general post, the messenger could be an impostor that gives the general contrary information.

In this scenario the question that arise is: *How do you get consensus from all the generals in order to attack or not attack at dawn?* [32] This was an open question for programmers and computer scientists until an anonymous source called *Satoshi Nakamoto* came up with a solution: *the blockchain* [33]. The solution for the Byzantine generals would involve each general having a notebook with the attack strategies written on it (to attack/not to attack). The notebook would have to be respected at all time and in case of modification, all generals would have to get into an unanimous consensus. This notebook can also be called *ledger*.

What does all of this has to do with a data marketplace system? By solving the Byzantine Generals Problem, blockchain allows for a communication protocol that works without a central authority managing everything. And because this thesis goal was to provide a two stakeholder trustable transaction without the presence of an intermediary, the blockchain technology is the right fit for it.

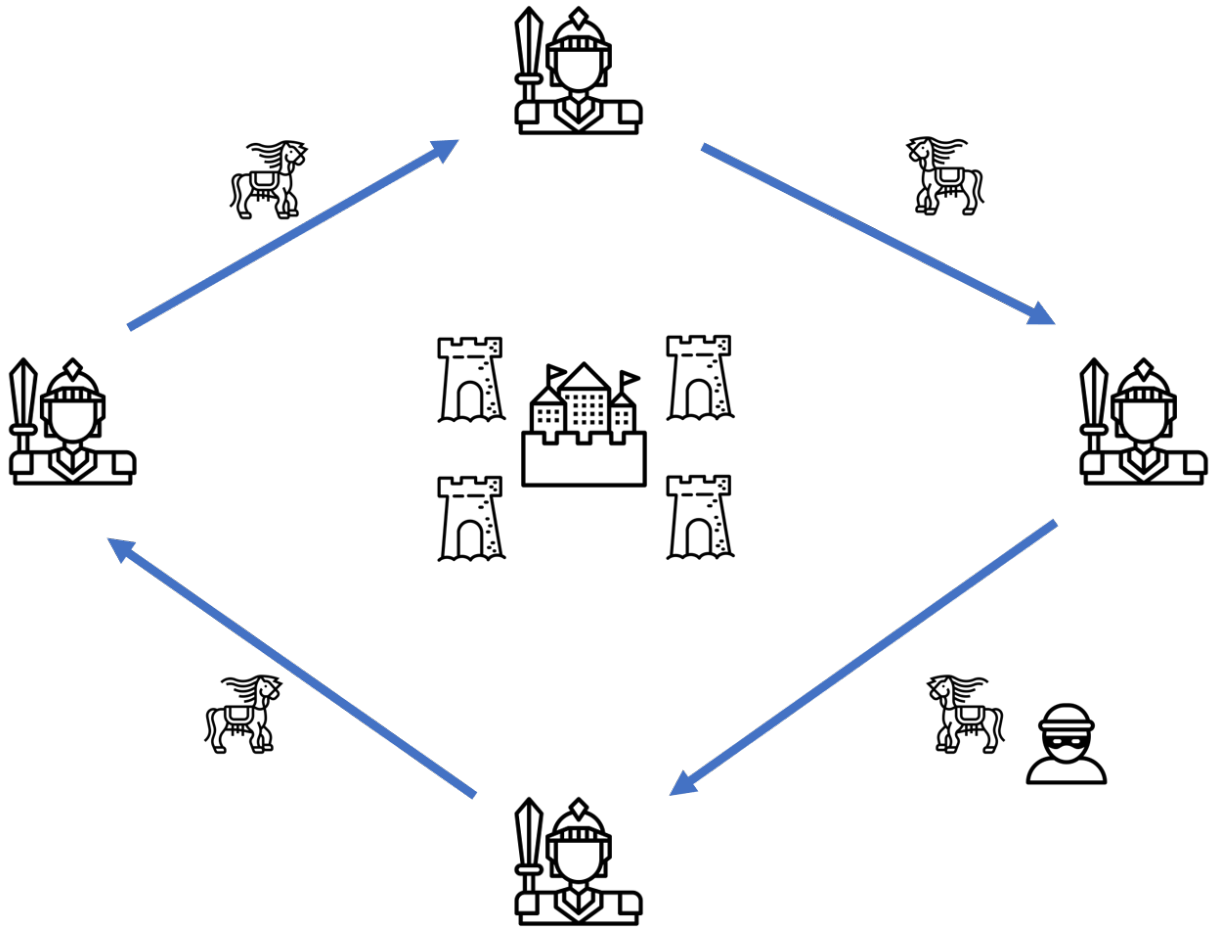

Figure 3.1: Byzantine Generals' problem

### 3.1.2 Blockchain

In the middle of the global financial crisis of 2008, an anonymous author called *Satoshi Nakamoto* proposed a new architecture for a peer-to-peer electronic cash system in order to avoid *the weaknesses of the trusted based model* used until then by all financial institutions (e.g. banks) [33]. His idea was to create a digital system in which non-reversible transactions could happen between two peers without the need of a trusted third-party intermediary.

The blockchain is a digital ledger that registers every transaction happening on the system (in this case, the data transactions). It is a set of consecutive blocks.

Each block is composed of a hash, a bunch of data and a signature (Figure 3.2). The hash is a long hexadecimal number intended to uniquely identify each block and its inner content. If the content changes, then the hash would change, showing that something is different from the registered block. The previous hash showed in Figure 3.2 refers to the hash of the previous block. The data is information about something (it could be money transactions, it could be data states, etc). The signature is the owner's proof to validate the use or modification of the data.

In order to avoid centralization and a single point of failure, the digital ledger is distributed across the people who support the system. These people can be divided in two:

nodes (computers) which only keep track of the registered transactions and miners (computers) who actively validate transactions therefore creating new blocks.

To validate each block, miners have to solve a mathematical puzzle as part of the consensus algorithm used. When a miner successfully solve this problem, it sends the answer to the others miners notifying them the puzzle completion. If the majority of miners is able to verify the solution, the block is added to the blockchain and the miner is rewarded with an amount of created bitcoin.

The validation process takes time to complete therefore each block contains multiple data or transaction registers instead of just one data register per block.

The new generated block is connected to the one before by adding the hash of the previous block. That is why the name of *block chain*.

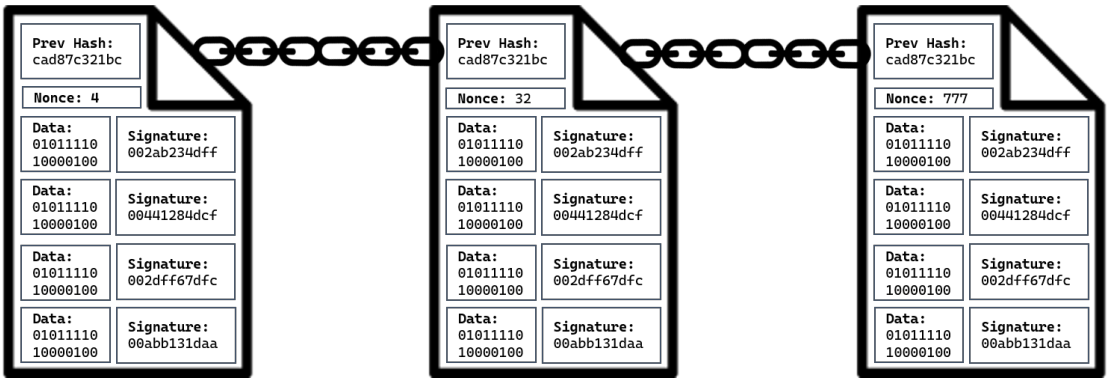


Figure 3.2: Block structure in the blockchain

The reason behind the *Decentralized Ledger* term comes up because the architecture distributes the ledger (blockchain) to multiple nodes (computers) connected to the system rather than having a single digital ledger on a centralized node.

Blockchain technology powers most of the current cryptocurrencies and some of them provide a platform for developers to use the decentralization feature in their applications by using special programs over the blockchain called *Smart Contracts*.

## 3.2 Types of Ledgers

There are different types of decentralized ledger technologies available at the market. The following section describes the most popular ones and their uses.

### 3.2.1 Bitcoin

The concept of Bitcoin appeared in 2008 in the same paper that presented blockchain for the first time [33]. Bitcoin can be considered the first application of blockchain technology by using a digital ledger where all bitcoin transactions are recorded and a consensus algorithm is used every ten minutes approximately to add a new block of transactions. Each participant (people who uses bitcoin) has a unique address which stores the participant's bitcoins.

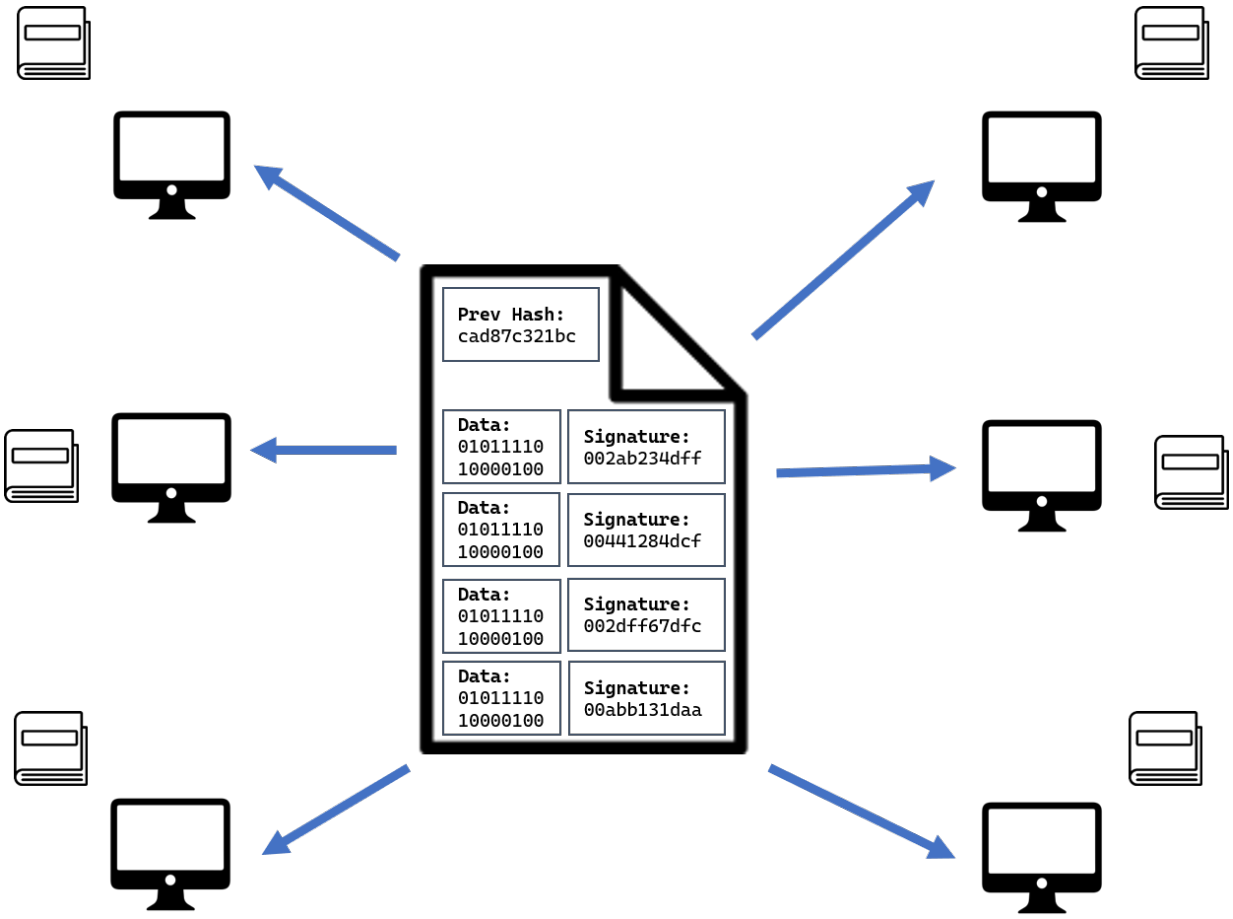


Figure 3.3: A decentralized ledger (blockchain) distributed along the participantes (nodes or miners)

The consensus algorithm used in Bitcoin is called *Proof of Work (PoW)* and is based on a computational challenge that is hard to get the answer but very easy to verify it. The basic idea behind it is:

1. A new set of transactions (block) needs to be added to the blockchain.
2. The information of the block (who is sending bitcoin, who is receiving bitcoin and how much is being sent), the hash of the previous block, the timestamp, the miner address account and a random value (called nonce) are concatenated and inserted into a hash function (in Bitcoin's case it is the SHA256).
3. The new generated hash needs to have at least $n$ zeroes at the beginning of the string with $n$ being a consensus criteria based on the amount of current miners and computer power in the system.
4. If the previous condition is not met, then the nonce value is changed and the hash generation process repeats again until a solution is found. This process is the *hard-to-solve* computational challenge of the PoW algorithm.

Figure 3.4: Bitcoin logo

5. The solution hash is easily verified by inserting the information of the block, the hash of the previous block, the timestamp, the miner address account and the nonce into the hash function. It needs to be verified by the other miners in order to reach consensus and approve the block creation in the network.

The problem with PoW is the computational resources wasted in solving the consensus challenges. At the time of writing, Bitcoin has been criticized for consuming more energy per year than Ireland [34].

Another issue with the Bitcoin protocol is the difficulty to write applications over its blockchain [35]. The Bitcoin protocol is specifically designed to be a peer-to-peer cash system rather than a platform for multiple and diverse applications.

### 3.2.2 Ethereum

Figure 3.5: Ethereum logo

After Bitcoin appearance, a young programmer called Vitalik Buterin saw the potential of the blockchain technology and the decentralization feature. That is why he started working on a blockchain protocol that would simulate a *quasi-Turing-complete* decentralized machine were programs could be executed. In 2015, the Ethereum project was finally released and since then has been growing in developers community and technical improvements [36].

As previously stated, Ethereum simulates a *quasi-Turing-complete* machine (called *Ethereum Virtual Machine* or *EVM*) in every node in the system that stores the state of the multiple programs executed at a given time. For explanation purposes imagine an object (like a Python dictionary or a JSON object) stored in a cloud server. Users can read

or modify its content with the right permissions. Users can also send cryptocurrency to this object. The caveat is that this cloud server is not a single machine but it is duplicated in the multiple nodes participating in the Ethereum network.

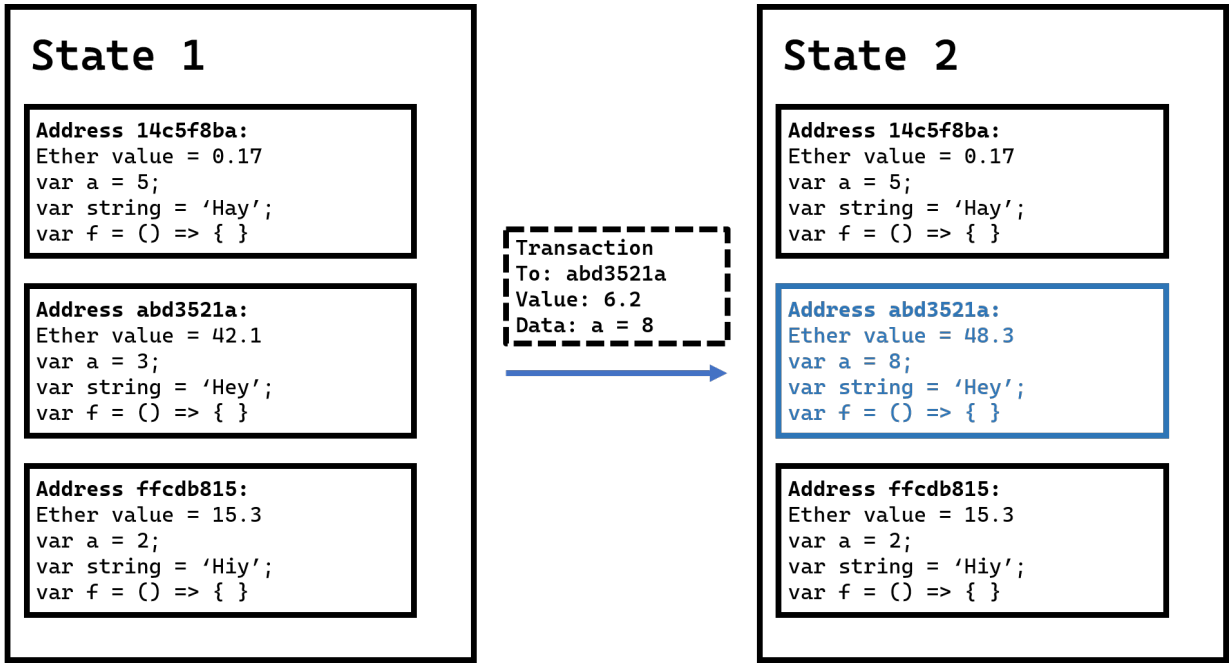


Figure 3.6: Change from state 1 to state 2 in the Ethereum's virtual machine

The Ethereum protocol currently uses PoW as its consensus algorithm but it is in the middle of the transition to a *Proof of Stake* algorithm which reduces the energy waste and makes the whole protocol more accessible for miners without large and expensive mining hardware.

This blockchain protocol was choosen for this thesis work because of its large developers community, extensive tutorials and promising growth opportunities. Nowadays it is the second most famous blockchain platform in the market and its constantly growing by improving its inner technology [37].

### 3.2.3 Hyperledger


Figure 3.7: Hyperledger logo

Blockchain protocols such as Bitcoin or Ethereum are called public blockchains because anybody can join the network and participate in the process as nodes or miners. For more

complex blockchain applications that deal with private or sensitive information, a public blockchain is not a viable solution because all information stored in the blockchain is public to all nodes and miners in order to properly validate and verify transactions.

Hyperledger is a set of open-source blockchain tools and frameworks hosted by the Linux Foundation to develop and deploy non-public blockchains called *permissioned blockchains*. Fabric is the main system of the Hyperledger set that allows entreprises to take advantages of the decentralization feature in *non-totally-trustable* business settings (e.g.: inter-corporations interactions, internal departments transactions, etc) without compromising the confidential information of its users [38].

A drawback of Hyperledger is that all nodes and participants have to be previously approved by a specific authority therefore making the decentralization feature a bit oscure. The previously described issue is the main reason why this blockchain technology was not selected for this thesis work.

### 3.2.4 IOTA


Figure 3.8: IOTA logo

IOTA uses another type of decentralized technology than blockchain, called *tangle*. The idea behind the tangle is that each user performing a transaction in the system has to first validate two previous transactions before registering his/her transaction. In this way, the ledgers looks more like a circular tangle with multiple available edges than a chain with a single available edge [39].

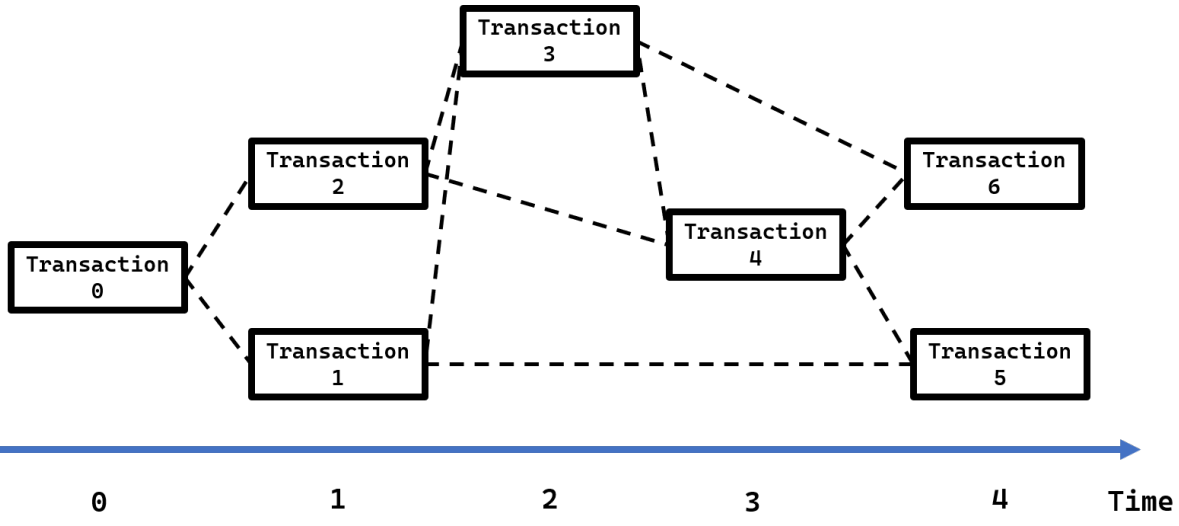


Figure 3.9: Tangle technology workflow

The tangle technology improves the energy consumption issue of the blockchain technology and also allows for a high latency and throughput for transactions.

The IOTA community is growing but has yet not created a strong documentation and extensive tutorials to use its platform. It was worth mentioning this cryptocurrency because of the related data marketplace system that the IOTA community has released and was considered as a possible tool for this thesis work.

## 3.3 Smart Contracts

### 3.3.1 Concept

As mentioned above, blockchain is a decentralized ledger that provides an underlying structure for applications to be run on top of it. These applications have one or more scripts that are executing on the blockchain and are called *smart contracts*. The term comes because the script is guaranteed to be executed after being successfully deployed in the blockchain infrastructure and it can be viewed itself as an agreement (or *contract*) that would be executed without the need of human supervision (*smart*) [40].

To sum up, a smart contract is a piece of code that can be executed on top of a blockchain. Although, blockchain developers made it easier to write smart contracts using high-level languages, there are some restrictions worth to be mentioned in the following subsection.

Most of smart contract compilers output the following files that are useful for deploying the smart contract on the blockchain and for interacting with it from any client node.

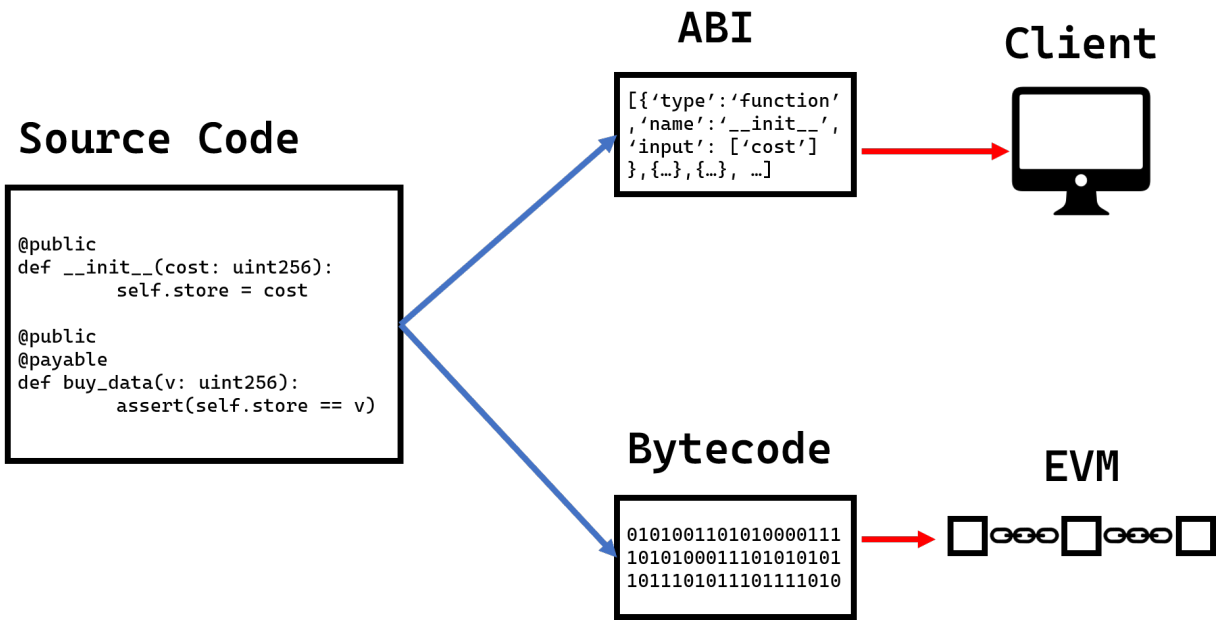


Figure 3.10: Results of compiling a smart contract source code

#### Bytecode

The smart contract bytecode is the raw *low-level* machine code that gets executed in the blockchain nodes when a smart contract is deployed. It contains all the smart contract data and the functions that are to be executed when needed or requested [40].

#### ABI

A smart contract Application Binary Interface (ABI) file specifies how to interact with the deployed smart contract. It is a set of declarations that specify what functions are available, what are the functions names and what inputs each function needs to be executed correctly [36].

### 3.3.2 Programming Languages

On the Bitcoin era, it was possible to develop a smart contract on top of the Bitcoin blockchain but the developer had to understand the low-level language *Bitcoin Script* that interacted with the blockchain in order to register transactions [35]. The Ethereum era brought with it a new paradigm focused on decentralized application and multiple high level languages appeared to help developers deploy smart contracts in a secure and fast way [36]. Today, there is a diverse group of smart contracts programming languages for the diverse blockchain infrastructure it spans (Ethereum, Hyperledger, Cardano, etc). For this thesis, the two most used and famous programming languages in the Ethereum space were considered:

- **Solidity:** is a *Javascript-like* programming language. Nowadays it is the most used programming language in the Ethereum space thanks to its flexibility, *low-level* interaction and accessibility with the blockchain. The developers using Solidity can use all the data types available and create complex procedures to be executed [41]. This feature has allowed experienced programmers to build high level applications with complex functioning but also has allowed critical bugs that have ended in catastrophic situations [42].
- **Vyper:** is a *Python-like* programming language that was created to implement the Python philosophy principles such as simplicity and readability [43]. This language was selected to build the smart contract of this thesis work because it allowed a quick learning of its functioning and prevents identified bugs that other languages do not prevent.

The description and code of the smart contract implemented in this thesis work is detailed in the Methodology section.

### 3.3.3 Challenges

When deploying smart contracts into the blockchain, various challenges appear. The developers have to be aware of them in order to build secure, functional and efficient decentralized applications.

### Limited Patching

The immutability feature of the blockchain creates a barrier for developers to patch their code in case of bugs and errors. This is caused because after deploying a smart contract into the blockchain, it is impossible to make any changes on it that were not previously foreseen. This challenge is *not a bug but a feature* but still causes problems to programmers who have to think more deeply about their code and vulnerabilities before deploying their smart contracts.

To solve this issue, multiple smart contract analysis tools such as Smartcheck [42] have appeared in the space. These tools allow developers to identify critical bugs on their code but they would not automatically identify them all. Another solution is a testnet (testing network) available to deploy and test smart contracts. These tesnets simulate the real blockchain networks but with fictional smart contracts and cryptocurrencies [40].

For this thesis work, the Ropsten testnet was used to test and deploy the smart contract of the data marketplace. The network allows for multiple tries and testing without spending real cryptocurrency in the process of deploying and interacting with the smart contract.

### Privacy Concerns

The decentralization feature of the blockchain allows anyone to check and validate all interactions and transactions happening on the network. Even though this open auditability guarantees the correct execution of the smart contracts, it also creates a privacy challenge for sensitive information sent to and stored on the blockchain.

In order to mitigate the privacy issues that could come up in a decentralized application, developers have opted to use the *pseudo-anonymous* nature of the users address. This means that any user can privately run or have sensitive information on the blockchain if nobody knows which user address it is using. So a user who wants to perform a private transaction can use a different address than the one he or she usually uses.

Another way to solve privacy concerns is to create application architectures that make the private computations or sensitive information processing on the clients and only send a final encrypted, hashed or obscured result to the blockchain to register the operation.

There is extensive bibliography which addresses the smart contract privacy concerns challenges and multiple solutions that have arose like Zether [44] and Hawk [45].

### Gas Restriction

The smart contracts deployed on a blockchain have to be executed by all the network's nodes in order to have a *synchronized state machine* that hold the same states in all nodes and therefore hold the correct state of the blockchain. The gas concept was introduced to avoid infinite smart contracts executions that could cause the nodes to run indefinitely and therefore take down the system [36].

To run any smart contract operation on the blockchain, developers and users send a cryptocurrency amount to the system that is distributed to the network's nodes performing the operation. If an interaction runs out of gas, the execution is stopped and reversed (depending on the blockchain system). This way the network's nodes are guaranteed that all smart contract execution would be finite and they would be rewarded for it.

An important feature of the Vyper language that prevents developers to write code that halt at a specific time, is that Vyper does not have loop operations (such as *for* or *while*) [43].

### Node Connectivity

The blockchain is a network of independent nodes that run and register all operations performed by the smart contracts and therefore the applications using that blockchain. In order to communicate with it, the user would have to have a running node that can at least read and send requests/operations. In the Ethereum blockchain, these nodes are called *Ethereum Clients* and sometimes required a moderate computer knowledge to set it up.

A company called *Infura* [46] provides a fast solution for developers to avoid having to set up their own nodes and instead use one of the *Infura* nodes already connected to the blockchain. Their services abstract away all the infrastructure and node management tasks for developers to help them focus on their application core logic. The company has a free plan that allows:

- Ethereum Mainnet and Testnets access
- 100,000 blockchain requests per day
- 3 different projects
- Community support forum

# Chapter 4

# Related Works

There are multiple data marketplaces solutions in the market nowadays. The following list of related data marketplaces systems has been selected because of its similarities with the system proposed in this thesis and its reputation. It should be pointed out that there are multiple academic and theoretical proposals for a data marketplace system in the related bibliography [47][48][49][50] that have not made it into the list because of its early development and *proof-of-concept* phases.

## 4.1 IOTA Data Marketplace


Figure 4.1: IOTA Data Marketplace Advertisement

In 2017, the IOTA foundation launched the first Data Marketplace platform using the tangle technology described in the *Blockchain* chapter. From then until now they have

more than 70 organizations signed up and they are still in the test phase of their system according to their website [51].

The main objective of the platform is to provide a way for organizations and individuals to offer their data (usually generated by IoT hardware such as Smart Fridges, Smart TVs, sensors, etc) to any stakeholder interested on it. The platform has a transparent and *no-third-party* mechanism to perform the transaction allowing the seller to receive compensation for his/her data contribution while the buyer acquires access to more data for his/her purposes. The functioning is the following:

1. The seller prepares a web server with an API that would authenticate requests and stream the data if they are accepted using a *Mask Authenticated Messaging* protocol to encrypt the communication layer.

2. The buyer acquires the data using IOTA cryptocurrency and is able to read the encrypted data stream of the seller.

3. The buyer perform the desired operation with the acquired data.

In this platform, the data received by the buyer is unencrypted so a malicious buyer could buy the data one time and sell it later at a lower price in unlimited occasions. It is worth nothing that because the data is usually coming from IoT devices, then chunks of data might not be worth enough for potential new buyers so the malicious buyer would have to continuously acquire data from the seller.

The IOTA Data Marketplace is still a prototype and further development is needed in order to offer a scalable and reliable solution for the data industry.

## 4.2 Ocean Protocol

Ocean Protocol is a set of open-source tools to deploy data marketplaces that use *data tokens* based on the *ERC-20 tokens* that simulate a cryptocurrency environment and transactions on the Ethereum network [52].

The foundation who built it was aiming for a data marketplace system where users could commercialize their data while maintaining its control (totally opposite of today's *Big-Tech* companies that collect user data, profit from it and do not distribute to its owners).

The set contains multiple tools such as an *Automated Market Marker* to set prices for the offered data sets, a *Compute-to-Data* feature that performs computations on private data in the owner premises and send the result to the buyer, and others [52].

A huge challenge for the *Compute-to-Data* feature is the integrity of the computation in the *On-premises* data servers. The buyer does not have a guarantee that its desired computation was performed correctly because a malicious data seller could have sabotage the execution. At the same time, the buyer does not have a guarantee that the offered data is correct and does not have considerable errors or noises. One of these challenges is addressed in the Methodology section while the other is left for future work.

Even though the Ocean Protocol community is growing at a fast rate, the documentation for understanding and implementing solutions was not clear enough to be used in this thesis work.

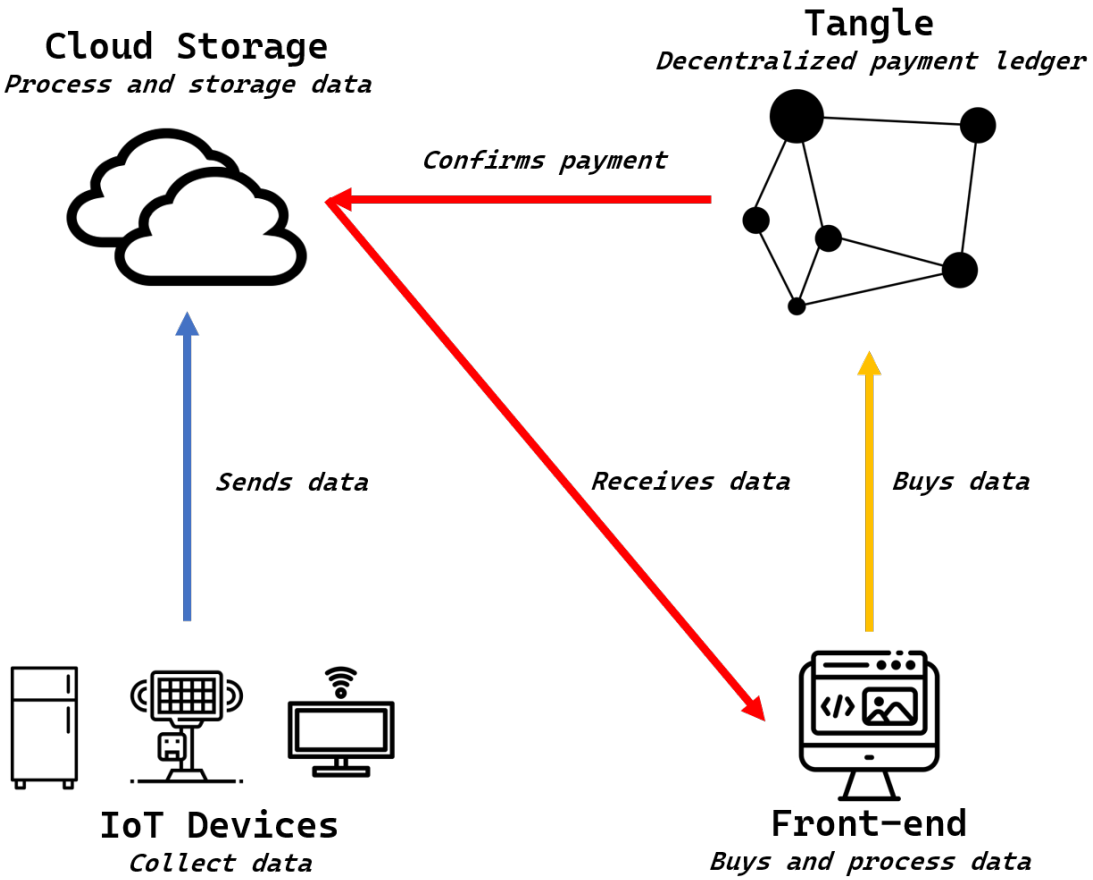


Figure 4.2: IOTA Data Marketplace workflow

## 4.3 Sterling

The Oasis Labs was cofounded by a PhD. Dawn Song, a Computer Science professor at UC Berkeley. The goal of the labs is to provide tools for *controlling and sharing data*. The Oasis Blockchain Platform [53] implements *trusted-execution-environment* (TEE) technology to guarantee data privacy in all operations performed on the platform. This blockchain allows developers to built multiple applications on top of it, one of them directly relates to the goals of this thesis work: Sterling [54].

Sterling is a data marketplace that uses the Oasis blockchain in order to guarantee data privacy for its users. The buyer pays for the data access which is sent to the blockchain at the same time the buyer sends the algorithm to proccess it. The TEE blockchain nodes input the encrypted data, its decryption key and the processing algorithm in the TEE, perform the computation and output the final result to the buyer. In the whole process, the TEE enclave nature does not allow the blockchain nodes and neither the buyer to see the decrypted data [54].

The Sterling data marketplace security depends on the security of the TEE enclaves that requires its blockchain nodes to have before joining the network. It is a *hardware-based* solution for the data privacy problems described in previous sections.

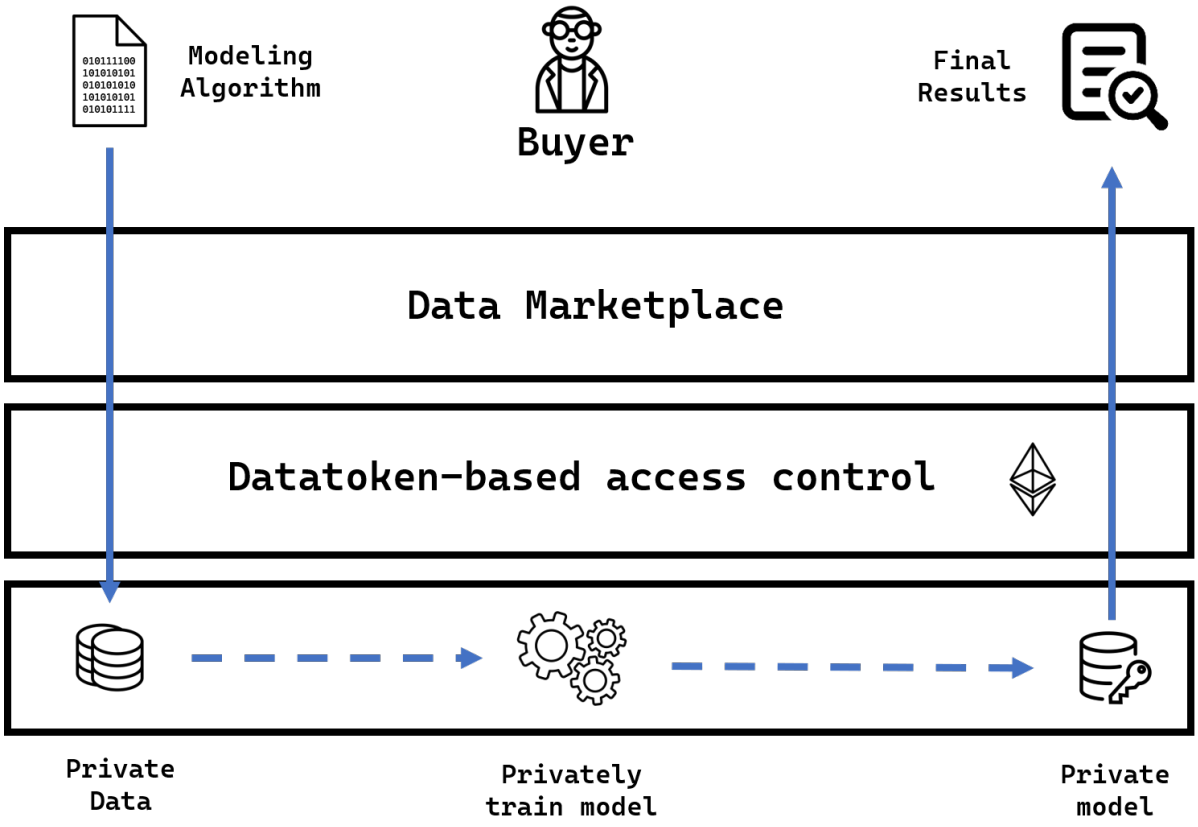


Figure 4.3: Ocean Protocol workflow

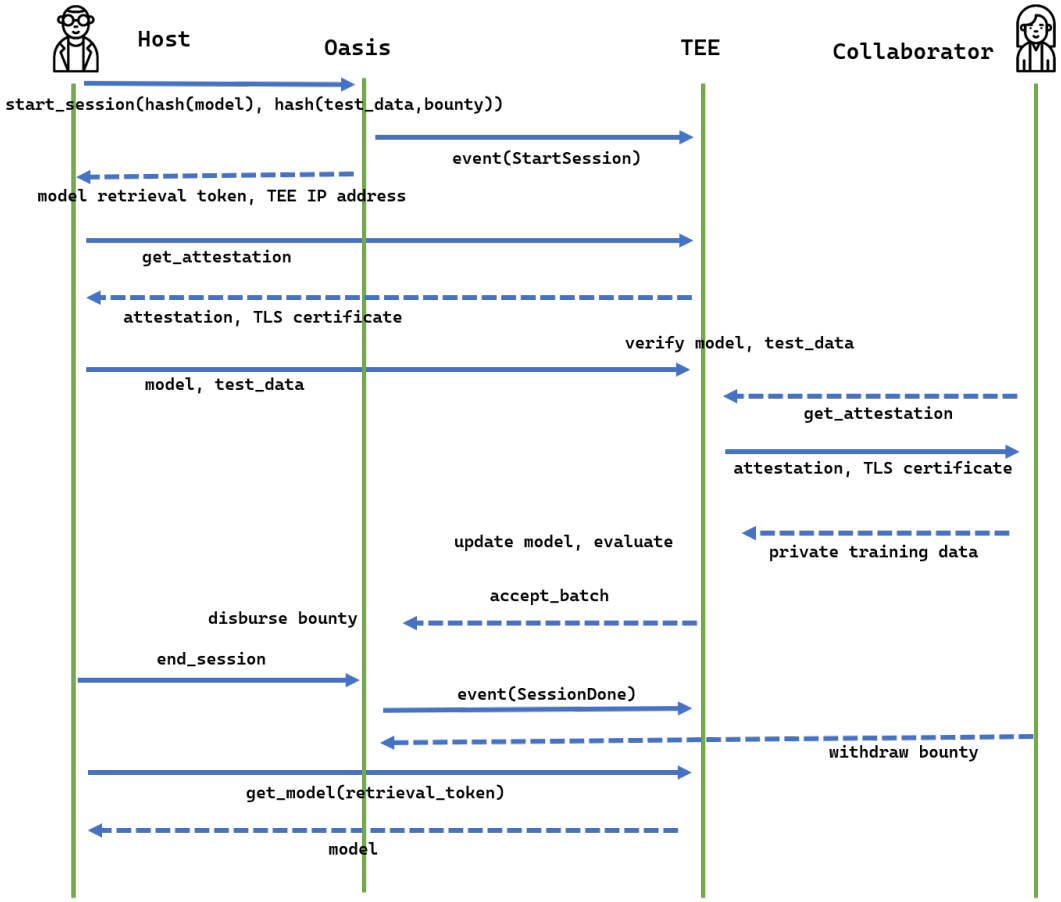


Figure 4.4: Sterling Data Marketplace workflow

# Chapter 5

# Methodology

This chapter describes the implementation of the data marketplace system proposed in this thesis work. It combines homomorphic encryption and decentralized ledger; the first is to avoid data leakage by maintaining the result when operating over encrypted data; the second, to guarantee a correct purchase process without intermediaries. The system is divided in two main parts: the seller *(server)* and the buyer *(client)*.

The buyer buys data from the seller and sends the payment to the decentralized ledger which is acting as an intermediary. The seller sends encrypted data to the buyer who would compute it in her own machine. After finishing the process, the buyer sends the encrypted result to the buyer for decryption. The seller sends the decrypted result to the buyer and request his payment to the decentralized ledger.

The code has been open-sourced by the author and can be found at the Github Repository: `https://github.com/NicoSerranoP/HEsystem` or can be directly installed using `pip` as described in the later chapter.

## 5.1 System Architecture

The data marketplace system is based on a *client-server* architecture where the seller or data provider would set up a web server to continuously offer its data and multiple buyers would be able to request test data and real data to run their private algorithms in their local machines. The three main phases are:

1. Request encrypted data
2. Perform computations over encrypted data
3. Request final result

Note that the client has to perform two request to the server (to ask for encrypted data and to ask for the decrypted result). This is because the server is the only one with the encryption keys.

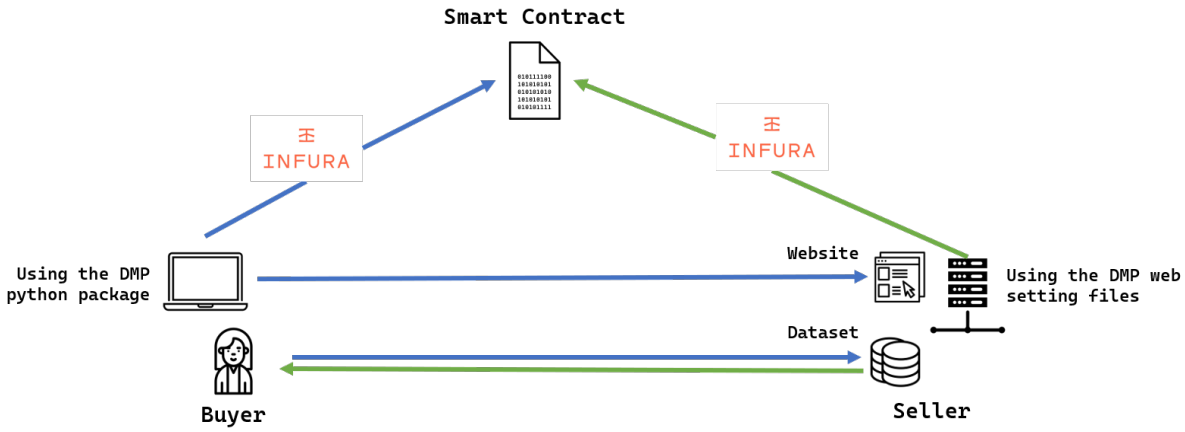


Figure 5.1: The data marketplace (DMP) system architecture

### 5.1.1 Payment

The payment process would be controlled by a decentralized ledger (a.k.a. smart contract) which would initially receive the buyer's payment and would track every step of the transaction. If all steps are completed correctly, the smart contract would send the buyer's payment to the seller. The buyer and the seller would use the *Infura* service to communicate with the decentralized ledgers. It is possible to set a direct communication channel between the stakeholders and the decentralized ledgers but it is not in the scope of this work.

Each data set is assocciated with a smart contract, which includes the price, the guarantee value, the data APIs links and the defined functions to register buyer and seller interactions. This helps organize the interactions between sellers and buyers. For example a buyer would be able to buy multiple data sets at the same time from the same seller without confusing the decentralized ledger with the transaction's steps.

If the seller wants to offer different portions of the data set, the seller would have to consider each portion as a new data set therefore creating a new smart contract and providing new APIs for interactions.

### 5.1.2 Workflow

The workflow of a given transaction between a seller and a buyer would be the following:

1. Seller sets a web server which showcase a simple website with all the required instructions for potential buyers to follow (an example HTML website is provided to modify it according to the seller specifications). This is an *once-in-a-lifetime* event.

2. Buyer visits the seller's website to learn about the available data and its costs. In this website the buyer can also find technical instructions for fetching the encrypted data.

3. Buyer sets up a Python environment with the data marketplace system package installed.

4. Buyer can access to a small data sample (for free) or to a large data sample with particular buyer-defined features (for a payment) through the package functions.

5. Normally, the buyer first wants to test her back-end program with a small data sample. For this, he can access for free to a small data sample by using the appropriate APIs.

6. After checking her back-end program runs correctly with test data, the buyer has to change the code so that it can now fetch real data. This also involves a payment, which depends on the data price indicated in the informative website of the first step.

7. Buyer runs the code which performs the following steps:

   (a) Ask for the seller's web API address (HTTP url).

   (b) Retrieves the smart contract information linked to the requested data.

   (c) Pays the data price to the smart contract.

   (d) Ask the web API to retrieve data

   (e) The web API checks the smart contract status to confirm whether it is safe to send data to the buyer. If confirmed, the web API sends the data to the buyer.

   (f) The buyer's code processes the data and sends the encrypted result to the seller's web.

   (g) The seller's web checks that the results follows the specified standards and sends the decrypted result to the client.

   (h) The buyer receives the results

   (i) The smart contract checks that all the steps have been completed and deposits the seller the data price it received from the buyer steps before. Then it resets the buyer register to allow him/her to buy the data again.

8. Buyer receives the final results of her computation and the seller receives the data payment.

In case of a problem in the middle of the transaction process, the buyer can reclaim its payment and cancel the transaction. The smart contract would then check which steps have been completed and allow the refund only before the buyer receives the decrypted result. This helps to protect the seller from malicious buyers trying to trick him.

### 5.1.3 Result validation

When the buyer request for the result decryption, the seller has to validate it does not leak the data set. For example, imagine a malicious buyer that request encrypted data and ask for the decryption of the whole set. The buyer would be able to get the plain data to sell it or reuse it later.

In order to avoid this problem we exploit the fact that the data exchange is always based on arrays. First, the seller sends an array of encrypted data to the buyer; then, the buyer returns an array of encrypted results to the seller.

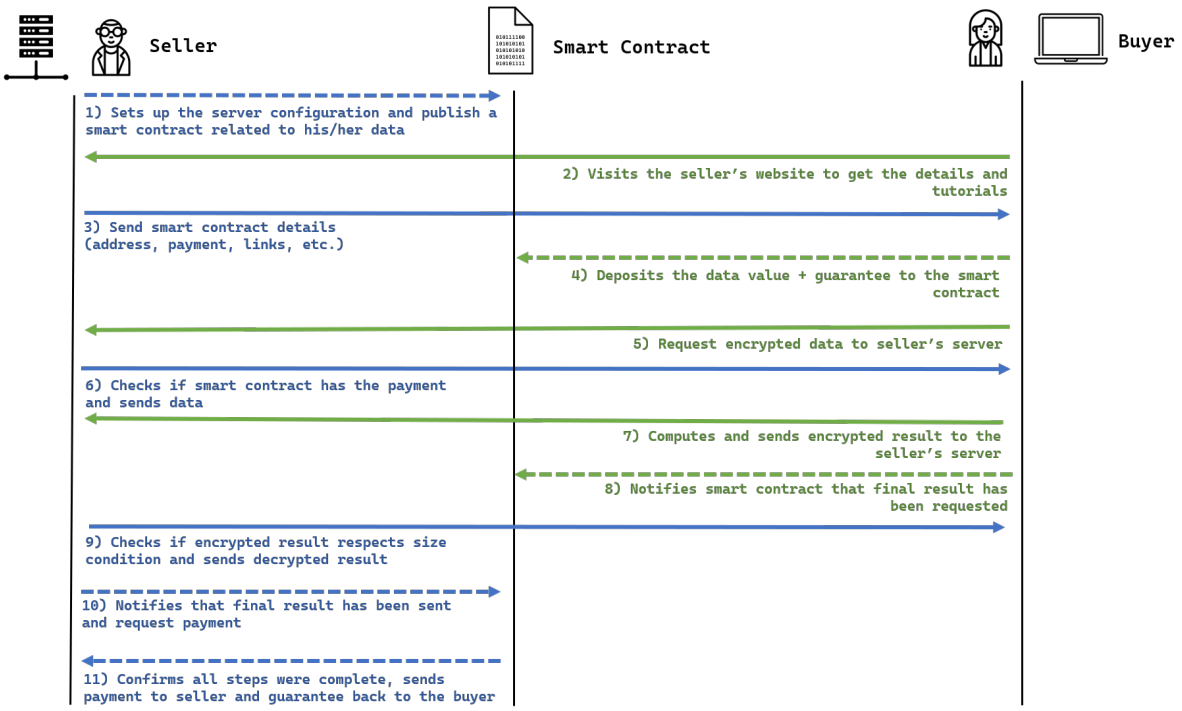


Figure 5.2: The data marketplace system workflow

In fact, our proposed anti-leakage validation consists on comparing both array sizes. If the size of the array sent by the buyer is larger than a secure threshold calculated dividing the seller array by a known constant, the decrypted result is not sent and the buyer is notified that the current solutions does not match the security parameters.

The cost of each data transaction prevents the scenario where the malicious buyer buys and decrypts multiple portions of the original data sets. For example, if the data set is 400 records long, the threshold is 10 and the cost of the transaction is $ 20 then the buyer would have to buy 40 times in order to get the complete plain data set. This means that the buyer would end up paying $800 and this scenario might not be worth it for him. It is noticeable that the security of the data set is directly proportional to the data set length. If the length is large enough then the scenario where a malicious buyer gets the whole plain data set is unlikely.

### 5.1.4 Target Users

The users (buyers and sellers) of the proposed data marketplace system would require medium to deep knowledge about computer science in order to use the python packages or set up the web server. Configuration instructions and tutorials are provided in the repository.

## 5.2 Used Hardware

The data marketplace system has been deployed and executed in a local machine with the following features:

- **Operating System:** Windows 10 Home
- **RAM Memory:** 16 GB
- **Processor:** Core i7 6th Generation 2.60Ghz (4 cores)
- **GPU:** Nvidia GeForce GTX 970M 3GB

The server application has been tested in a *Heroku* free-tier machine and it is designed to be used in any other hosting platform which allows the software requirements described below.

## 5.3 Software Requirements

In order to test and use the data marketplace system, both buyers and sellers have to meet the following software conditions:

1. **Programming Language:** Python 3.6+
2. **Dependencies:** Authorization to install python modules and libraries one by one or by installed all dependencies enumerated in the `requirements.txt` file in the repository.
3. **Package Installer:** The buyer user needs authorization to install the *data marketplace* module (`hesystem`) using the package installer `pip`.
4. **Dataset:** The seller needs a dataset on the server machine which could be fetched from a database application or by any file recognized by Python (CSV, Excel, JSON, etc). It is worth nothing that in order to use a dataset different than the default *Postgres* database configured in the server example, the user would have to change the proper changes to the application code.

# Chapter 6

# Results

This chapter discusses code examples to validate the correct functioning of the proposed data marketplace system. It shows the computation times required to run the examples and the pending challenges for the current data marketplace implementation.

## 6.1 Toy Examples

The data marketplace system developed in this thesis allows for multiple types of computations. The following examples were selected in order to prove that the data marketplace system works correctly but they are not the only operations in the system.

The first example is a mean calculation and the second one is a Machine Learning model training. The code for each one can be found at the `/tutorials` directory in the repository.

### 6.1.1 Calculating the average lifetime of a business

In the first example, a buyer, who expects to open a business downtown (for example a restaurant) is interested in knowing the average lifetime of the restaurants in a given area of the city in order to make a decision (for example how much to invest in his endeavor). To accomplish his goal, the buyer exploits the database of Innomaps, the seller. As mentioned before, Innomaps owns a database, which includes the locations, annual revenues, creation date and a wide assortment of business-related variables of many companies of Ecuador (and eventually the world).

In order to trade data, the buyer and Innomaps must play the roles of client and server in the proposed data marketplace system. Therefore, the buyer must install the client application in his local computer, whereas Innomaps has to do the same with the server application (see Annex B for detailed instructions). But that is not all. It is essential for the buyer to develop by itself a back-end program for processing the encrypted data received from the server. In this case, the back-end program must calculate the average of several inputs that will be sent from the server as an array of encrypted data.

The buyer has to check on an informative website previously set up by the seller in order to find the payment procedure and additional technical details such as the data requests

APIs, the test data requests APIs and the result requests APIs. To start the transaction, the buyer pays the data price to an intermediary (the smart contract) which would store the payment until the whole process is perform correctly. The smart contract tracks every possible state of the data trade, e.g. the buyer has paid, the seller has send encrypted data, the buyer has requested the final result, the seller has sent the decrypted result and the buyer has received the final decrypted result.

The back-end program starts by requesting data through an specific API. It does not have to decrypt the numbers received from the server. Whoever develops the back-end program can pretend that he is working with regular numbers, forgetting the fact that the numbers are encrypted. The average obtained by the back-end program can only be decrypted in a second data exchange with the server: the back-end program send the encrypted average to Innomaps, who decrypts it and send it back to the client. Only in that moment, the client gets what he wanted: the average lifetime of a restaurant, represented as a regular number. At the same time Innomaps notifies the smart contract that the transaction has been completed successfully and receives the payment. Since only Innomaps has the decryption keys, the client will never be able to see the raw data stored in the server.

The described workflow in Figure 6.1 allows both the buyer and Innomaps to enjoy benefits. Innomaps can breathe a sigh of relief; the client will never be able to see Innomaps real data; thus preventing him from reselling private data to other data seekers behind Innomaps' back. On the other hand, the buyer can perform any computation with the encrypted Innomaps' data, but Innomaps will never know what the buyer did; thus, the effort put into developing the back-end application can never be exploited by Innomaps.

In case that the buyer could not finish the transaction and did not receive the decrypted final result, he can start a litigation by notifying the smart contract. It will check whether the transaction failed, in which case it would repay the money to the buyer.

As it can be seen, the proposed data market system allows information extraction by maintaining data and computation separated into two: a client (the buyer) and a server (Innomaps). Homomorphic encryption makes the trick possible by maintaining the computations over encrypted data after decrypting the result.

Unfortunately, every benefit comes with a cost. For every operation with encrypted data performed on the client side, some noise is introduced in the result. This is due to the nature of Homomorphic Encryption that adds a small random noise every time it encrypts a number. The buyer has to consider his limitation when building his back-end program. Nowadays there is a new proposed solution to this problem called bootstrapping (and discussed in detail in Chapter 2) but the current used library has not implement it yet.

### 6.1.2 Predicting the average lifetime of a business

In the second example, a buyer, say an Artificial Intelligence startup, wants to create a tool to help business owners predict their business lifespan by analyzing some variables such as location, annual revenues, creation date among others.

The startup employees develop a basic logistic regression model which takes multiple attributes and predicts the average lifetime of a business. This model is considered the startup's intellectual property so they do not want it to share with anyone. Also the model

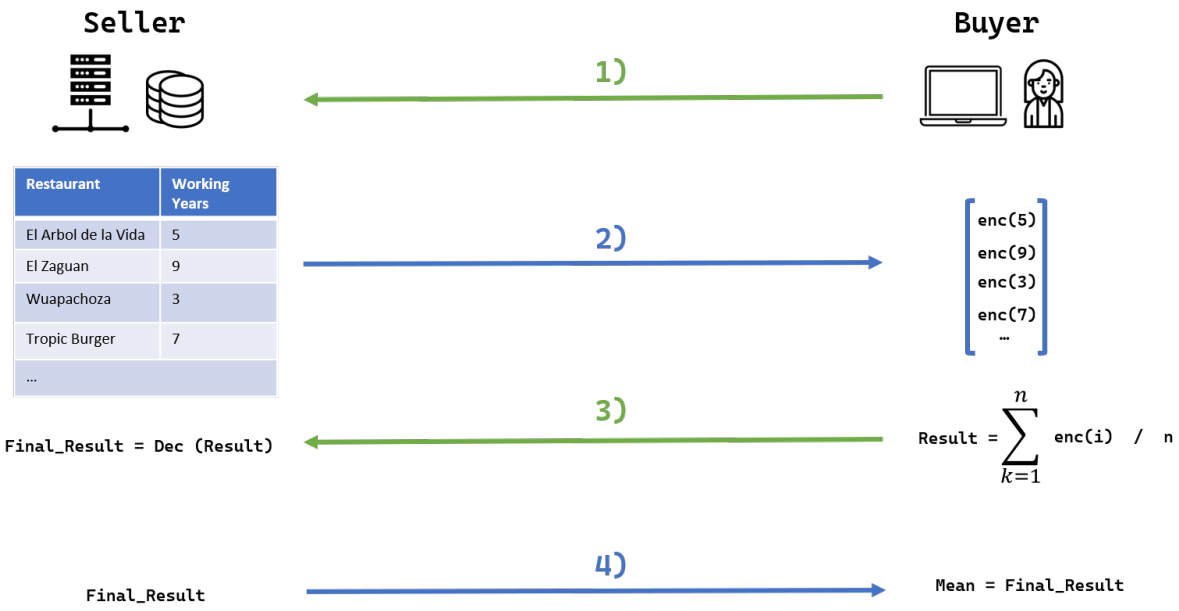


Figure 6.1: Mean calculation workflow

has been trained and tested with a small data set and therefore its overall accuracy may not have reach its maximum potential.

On the other hand, a seller, in this case Innomaps, has a considerable amount of data that would be useful for the startup to train their model.

The workflow that would allow the startup to train their model using Innomaps data set can be seen in Figure 6.3 and is described below.

The startup access Innomaps website and finds the right data set to feed their predicting model. As in the previous case, after paying for the data to an intermediary, a.k.a. small contract, the startup can request data to Innomaps through an adequate API from within a back-end program.

The back-end program will receive arrays of encrypted data from Innomaps. Each array will be used as an input to the logistic model (see Figure 6.2 a ). This means that each array will be multiplied by the model weights, add to the bias and passed through a sigmoid function.

After performing the training, the model would end up with the weights and the bias encrypted as indicated in Figure 6.2 b. This model is called *encrypted model*. As part of the payment, the startup has the right to ask Innomaps to decrypt the encrypted model. To do this, the back-end program has to send an array containing the weights plus the bias (all the data that needs to be decrypted) to Innomaps.

After receiving the decrypted result, the back-end program in the client would put it back into the model in order to obtain a non-encrypted model which would be useful for future predictions. At this time, the startup has been able to train their own logistic regression model using encrypted data from Innomaps and obtaining a ready-to-use, non-encrypted model without having to see Innomaps data.

From the foregoing, it seems that the buyer can try to steal Innomaps data by returning the same encrypted array to the server and asking for decryption. This is not possible.

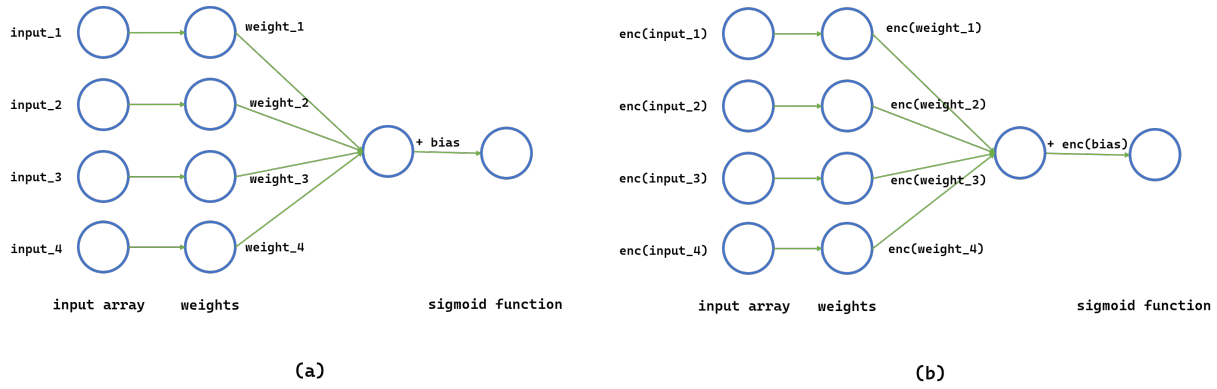


Figure 6.2: a) Logistic regression model. b) Encrypted model

The proposed datamarket place includes a simple but secure mechanism to identify such malicious attempts. The size of the array to be decrypted has to be a fraction of the size of the array sent initially to the client. The fraction is a parameter of the marketplace.

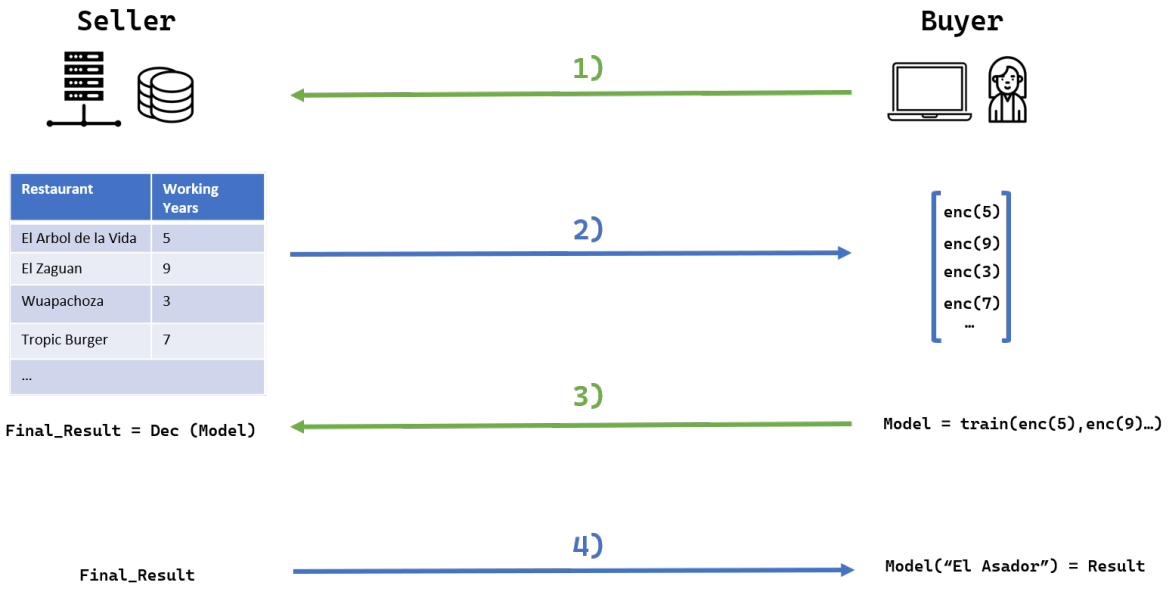


Figure 6.3: Model training workflow

## 6.2 Computation Times

The following table describes the computation times taken by the toy examples. Even though the homomorphic encryptions, decryptions and operations increase the computation time significantly, the privacy and data accessibility obtained by using this system allow for a broader use of private data.

| Operation | Time (seconds) |
|---|---|
| Plain vector sum | 0 |
| Encrypted vector sum | 0.153 |

Table 6.1: Computation times for calculating the average lifetime of a business (toy example 1).

| Operation | Time (seconds) |
|---|---|
| Plain model training | 0.002 |
| Encrypted model training | 119.91 |

Table 6.2: Computation times for predicting the average lifetime of a business (toy example 2).

# Chapter 7

# Conclusion

We are living in the era of data science and machine learning where data is becoming an extremely valuable asset. This asset needs to be processed before being useful for developers and researchers; the whole data's value chain involves work and resources therefore making the final result a costly item for the owner. With the data marketplace system proposed in this thesis work, the data owner would be able to offer his/her data for an amount of money, without compromising the privacy of the data set. The buyers are able to use data to extract information without seeing the raw data set thanks to homomorphic encryption. Unlike other data marketplace systems, this one involves only two participants (the buyer and the seller) without the need of an intermediary or third-party.

The data marketplace system has been proved with two different use cases and further development would improve the existing implementation, some specific open challenges and future work have been discussed in the previous chapter. All the system implementation is open-source and accessible for everyone to comment, use and improve while preserving their data sets privacy.

The answer to the question of **Who should share what?** is **No one** thanks to homomorphic encryption, distributed ledgers and this work.

## 7.1 Future Works

Nevertheless, the training phase comes with some challenges. The first one is that the startup developers have to find a way to make the sigmoid function work with encrypted number. This is solved by creating a function approximation using polynomials that would work over a defined range. In case the startup decides to change the model (for example to a neural network or a bayesian classifier) then they would have to approximate the corresponding functions again. The second challenge is that the noise introduced by the Homomorphic Encryption process could grow and affect the final result. In the current implementation, this challenge can be solved by implementing the right amount of operations or asking the seller to modify the encryption parameters that are used to create the encryption keys (such as polynomial length, coefficients, etc). In the future, this challenge could be overcome by using a method called bootstrapping, described in chapter 3.

The proposed data marketplace system comes with multiple challenges that can be a

stepping-stone for future works.

### 7.1.1 Ciphertext Growth

The homomorphic encryption scheme used in this system (CKKS) adds a small random noise to each number encryption. This noise will grow in each operation performed over the encrypted number, say an addition or a multiplication. As expected, some operations (like multiplication) would grow the noise faster than others (like addition). The noise growth implies that the underlying encrypted number (called ciphertext) also grows therefore generating a decrypted result different than expected.

All operations performed using the data marketplace system can increase the ciphertext length and therefore only a limited number of operations can be performed. To solve this challenge the seller could modify the initial encryption parameters (such as the polynomial size or the polynomial coefficient) in order to make each operation have less impact over the noise but making it more computational intensive. In this case, the seller would need a medium cryptography knowledge to set the right parameters.

A future solution is the implementation of the bootstrapping technique in the SEAL library. This method would decrypt and reencrypt the encrypted value in a given time in order to take out the noise. The raw value would not be visible to anyone thanks to an advanced play using two encryption keys in the same value. This solution is currently in the SEAL library road map[20].

### 7.1.2 Comparisons

As state by Rivest et al. in the first mention of homomorphic encryption [5] it is not possible to simultaneously preserve security and give the system the ability of performing comparisons against known constants.

For example, suppose that an homomorphic encryption scheme has the ability to compare encrypted values with known constants (numbers). An encrypted value could be 6 ($enc_val$). A malicious buyer could compare: $enc_val > 10$, the scheme would indicate $False$. Then the buyer would divide the known constant in half (5) and compare: $enc_val > 5$, the scheme would indicate $True$. The buyer would then change the comparison operator and add half of the known constant to the value: $enc_val < 7.5$, the scheme would indicate $True$. The buyer would substract half of the previously added half of the known constant: $enc_value < 6.25$, the scheme would indicate $True$. The buyer would continue the process until having a small enough interval that will indicate him the real value: 6.

At the beginning of this work a comparison method was proposed but after analyzing this challenge the proposal was discarded.

The alternative used for some comparison operations using this data marketplace system is function approximation. For example, the logistic regression model in the second toy example uses the sigmoid function to determine the outcome of the model in training. The function acts as a comparison operation because it determines which values are accepted and which ones are not given a threshold. The sigmoid function was approximated as a polynomial in a specific range using the *Stone-Weierstrass* theorem.

In case of changing the model, the user would have to approximate again the sigmoid function in the desired range to perform a correct training.

### 7.1.3 Data Visualization

Many times the buyer needs to know important properties of the data before start using it (e.g. its statistical distribuition, the presence of outliers, etc.) [55]. This is a problem because such properties cannot be determined from encrypted data.

A potential solution may be that the seller's server could have multiple APIs to ask for data with different specifications (normalized data with outliers, normalized data without outliers, standarized data with outliers, standarized data without outliers, etc). The problem with this solution is that the seller needs to built as many APIs as data combinations exists making it hard to maintain and scale.

A future work to solve this challenge would involve building data visualization tools that do not leak the data itself and inform the buyer about the data set characteristics.

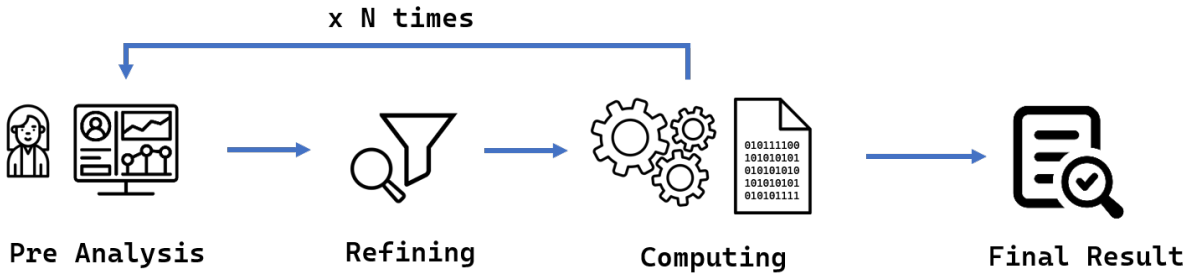


Figure 7.1: Process to create a ML model

# Bibliography


[1] M. Balazinska, B. Howe, and D. Suciu, "Data markets in the cloud: An opportunity for the database community," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1482–1485, 2011.

[2] F. Stahl, F. Schomm, G. Vossen, and L. Vomfell, "A classification framework for data marketplaces," *Vietnam Journal of Computer Science*, vol. 3, no. 3, pp. 137–143, 2016.

[3] Stack Overflow, "Developer survey," Stack Overflow, Tech. Rep., feb 2020, available at https://insights.stackoverflow.com/survey/2020.

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[5] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[6] C. Gentry, "A fully homomorphic encryption scheme http://crypto. stanford. edu/craig/," Ph.D. dissertation, thesis. pdf, 2009.

[7] D. S. Dummit and R. M. Foote, *Abstract Algebra*. Wiley Hoboken, 2004, vol. 3.

[8] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[9] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[10] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[11] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987. [Online]. Available: http://www.jstor.org/stable/2007884

[12] R. A. Perlner and D. A. Cooper, "Quantum resistant public key cryptography: a survey," in *Proceedings of the 8th Symposium on Identity and Trust on the Internet*, 2009, pp. 85–93.

[13] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–35, 2018.

[14] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[15] C. Gentry, "Computing arbitrary functions of encrypted data," *Communications of the ACM*, vol. 53, no. 3, pp. 97–105, 2010.

[16] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[17] D. Huynh, "Cryptotree: fast and accurate predictions on encrypted structured data," *arXiv preprint arXiv:2006.08299*, 2020.

[18] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23, easy-to-understand blogs can be found in https://blog.openmined.org/ckks-explained-part-1-simple-encoding-and-decoding/.

[19] ——, "A toolkit for ring-lwe cryptography," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 35–54.

[20] H. C hen, K. Laine, and R. Player, "Simple encrypted arithmetic library-seal v2. 1," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 3–18.

[21] Microsoft. (2017) Seal repository. [Online]. Available: https://github.com/microsoft/SEAL/tree/master

[22] O. Mined. (2020, jul) Tenseal. [Online]. Available: https://github.com/OpenMined/TenSEAL

[23] M. Egorov, M. Wilkison, and D. Nuñez, "Nucypher kms: Decentralized key management system," *arXiv preprint arXiv:1707.06140*, 2017.

[24] NuCypher. (2019, feb) A gpu implementation of fully homomorphic encryption on torus. [Online]. Available: https://github.com/nucypher/nufhe/releases

[25] F. A. R. Division. (2019, oct) Crypten: A new research tool for secure machine learning with pytorch. [Online]. Available: https://ai.facebook.com/blog/crypten-a-new-research-tool-for-secure-machine-learning-with-pytorch/

[26] ——. (2019, oct) Crypten repository. [Online]. Available: https://github.com/facebookresearch/crypten

[27] S. Halevi and V. Shoup, "Algorithms in helib," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 554–571.

[28] L. De Branges, "The stone-weierstrass theorem," *Proceedings of the American Mathematical Society*, vol. 10, no. 5, pp. 822–824, 1959.

[29] M. M. Waldrop, "More than moore," *Nature*, vol. 530, no. 7589, pp. 144–148, 2016.

[30] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[31] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[32] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 203–226.

[33] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Bitcoin, Tech. Rep., 2019.

[34] M. J. Krause and T. Tolaymat, "Quantification of energy and carbon costs for mining cryptocurrencies," *Nature Sustainability*, vol. 1, no. 11, pp. 711–718, 2018.

[35] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.

[36] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[37] P. Fairley, "Ethereum will cut back its absurd energy use," *IEEE spectrum*, vol. 56, no. 1, pp. 29–32, 2018.

[38] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[39] B. Kusmierz, "The first glance at the simulation of the tangle: discrete model," *IOTA Found. WhitePaper*, pp. 1–10, 2017.

[40] E. Foundation. Ethereum online documentation. [Online]. Available: https://ethereum.org/en/developers/docs/

[41] C. Dannen, *Introducing Ethereum and solidity*. Springer, 2017, vol. 1.

[42] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, 2018, pp. 9–16.

[43] V. E. Team. (2018) Vyper repository. [Online]. Available: https://github.com/vyperlang/vyper

[44] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 423–443.

[45] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 839–858.

[46] (2020) Infura. [Online]. Available: https://infura.io/

[47] K. Mišura and M. Žagar, "Data marketplace for internet of things," in *2016 International Conference on Smart Systems and Technologies (SST)*, 2016, pp. 255–260.

[48] G. S. Ramachandran, R. Radhakrishnan, and B. Krishnamachari, "Towards a decentralized data marketplace for smart cities," in *2018 IEEE International Smart Cities Conference (ISC2)*, 2018, pp. 1–8.

[49] K. R. Özyilmaz, M. Doğan, and A. Yurdakul, "Idmob: Iot data marketplace on blockchain," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 11–19.

[50] M. Travizano, C. Sarraute, M. Dolata, A. M. French, and H. Treiblmaier, *Wibson: A Case Study of a Decentralized, Privacy-Preserving Data Marketplace*. Cham: Springer International Publishing, 2020, pp. 149–170. [Online]. Available: https://doi.org/10.1007/978-3-030-44337-5_8

[51] I. Foundation. (2017) IOTA data marketplace website. [Online]. Available: https://data.iota.org/

[52] T. O. P. Foundation. (2018) Ocean protocol whitepaper. [Online]. Available: https://oceanprotocol.com/technology/roadmap#papers

[53] O. Labs. (2020) The oasis blockchain platform. [Online]. Available: https://oasisprotocol.org/papers

[54] N. Hynes, D. Dao, D. Yan, R. Cheng, and D. Song, "A demonstration of sterling: a privacy-preserving data marketplace," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 2086–2089, 2018.

[55] J. W. Tukey, *Exploratory data analysis*. Reading, MA, 1977, vol. 2.

[56] T. E. Foundation. (2015, feb) Web3 documentation. [Online]. Available: https://web3js.readthedocs.io/en/v1.3.0/

# Appendices

# Appendix A

# Glossary

- **Data marketplace system:** Refers to the system being developed in this thesis if not explicit indicating otherwise.
- **Stakeholder:** Refers to a person, group of people, organization or business. In other words: *an entity involved in the transaction*
- **Seller:** Refers to the stakeholder who is offering the data
- **Buyer:** Refers to the stakeholder who is acquiring or requesting the data
- **Private data:** Refers to a confidential, private or sensitive data set. Usually would be the data set involved in the transaction
- **Data set:** Refers to a structured set of data. Usually has a table structure
- **Encrypted data:** Refers to the hidden/encrypted data set
- **Decentralized Applications:** Refer to applications that do not rely on a centralized server to offer its services or products. P2P systems, blockchain applications and others count as Decentralized Applications.
- **Distributed ledger:** Refers to a virtual notebook that keeps the transaction status and can be accessed by any of the stakeholders. It is hosted in the blockchain
- **Plaintext:** Data that has not been encrypted (e.g.: original data)
- **Ciphertext:** Data that has been encrypted (e.g.: encrypted data)
- **Ring:** An algebraic structure that contains a set and two arithmetic operations (addition and multiplication)
- **Cryptosystem:** A set of algorithms to provide a particular service. They usually include a key generation algorithm, encryption algorithm and decryption algorithm.
- **Web3:** Is a collection of libraries that allow you to interact with a local or remote ethereum node using HTTP, IPC or WebSocket [56].

- **API:** An *Application Programming Interface* (API) is an interface to interact with a specific software or server.

# Appendix B

# Installation

In order to install and use the data marketplace system, two stakeholders need to be involve. One should act as the *data seller* (following the *Server Installation* instructions) and the other one should act as the *data buyer* (following the *Client Installation* instructions). It is worth mentioning that a stakeholder can be a seller and a buyer at the same time by installing the tool-kits in different machines.

## B.1 Server Installation

The following steps have to be taken in order to install the web app in with the informative website and the data APIs for the seller's server:

1. Open the terminal and move to the desired directory to be installed.
2. Download the server repository from `https://github.com/NicoSerranoP/HEsystem`.
3. Change the data details and urls in the `/templates/data.html` file according to the seller's specific information.
4. Create an `/info/user_info.txt` file with the user information such as address, private key and endpoint url
5. Create an `/info/contract_info.txt` file with the contract details such as meta link, test link, test result link, data link, result link, num rows, value, expiration time and condition.
6. Install the dependencies in the `requirements.txt` file by executing `pip install -r requireme`
7. Set up the environment variables for the *Postgres* database connection by executing `set DATABASE_URL=postgresql://postgres:postgres@localhost:5432/basic-provider` (Windows) or `$env:DATABASE_URL=postgresql://postgres:postgres@localhost:5432/bas` (Linux).
8. Start the web server by executing `flask run`

Some file modifications in the `app.py` file or the HTML files in the `templates` directory are needed in order to use different settings that the default ones (ex: *Postgres* database, *Flask* web server, etc).

The repository is configured to be run in a *Heroku* free-tier plan but it can be modified to be run in any cloud or hosting platform with the hardware and software requeriments described in the methodology section.

## B.2 Client Installation

The following steps are required to install the tools to interact with a seller's data marketplace system from a *Python* code.

1. Create a *Python* virtual environment to avoid confusion with other packages and dependencies by executing `python -m venv environment_name` and activate it by executing `cd environment_name/Scripts/ && ./ activate`
2. Install the *hesystem* package by executing `pip install hesystem`
3. Use one of the *.py* files in the `/tutorials` directory to understand how to operate the encrypted data