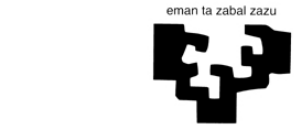

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Privacy-preserving protocol from Blockchain and Homomorphic Encryption

Final Master Dissertation
Master Tecnología Blockchain y Criptoeconomía

Leire Etxebarria Ikazuriagagoitia

Supervisor:
Albert Garreta-Fontelles
Oscar Lage Serrano

Leioa, September 18, 2021

Contents

Abstract

The advent of blockchain technology and the emergence of smart contracts have been a technological breakthrough in various sectors of society. The possibility of being able to carry out the verification, registration and coordination of transactions automatically and autonomously without the need for a trusted third party is a great advantage over traditional transaction systems. The correct application of this technology brings transparency, immutability and decentralisation to the processes that make up the activity of the scope of application, improving them significantly. Although blockchain offers a very expressive functionality, it does not provide privacy out of the box and although several attempts have been made in recent years to provide it with privacy, it has proved to be a challenge. In this final master dissertation we theoretically propose a protocol to create decentralised application for the Ethereum platform, equipped with a fully homomorphic encryption scheme, which allows computation with the data of different users while preserving the privacy of the data, the algorithm used and the answer obtained therefrom, at all times.

Objectives

The main objective of this work is to carry out the theoretical development of a protocol that allows the creation of decentralised application in which blockchain technology is applied together with fully homomorphic encryption. Specifically, the aim is to design a decentralised application in which some data owners can leave the data encrypted with a homomorphic scheme to a third party algorithm owner, so that the algorithm owner can apply the algorithm on the data. This is carried out in a way that the data is not disclosed at any time in the clear and only the result of the algorithm is revealed to the owner of the algorithms. For this purpose we have studied both blockchain technology and homomorphic encryption, and in Sections 2 and 3, respectively, we have written the history and the basic fundamentals of these technologies. In Section 4 we have written the obstacles we have encountered in the design of our protocol and the theoretical solution we propose. In Section 5 we detail the flow of our protocol and a proposal of the smart contract. The use cases in which this protocol could be used can be found in Section 6. Finally, in Section 7 we make a general conclusion of this work. All this starts with the introduction in Section 1, where we justify this work and our choice of technologies, and we explain the general idea of our application.

Chapter 1

Introduction

1.1 Motivation and justification of this work

In 2013 IBM said that %90 of the world's data at that time had been created in the last two years [1]. In the era of Big Data, data is constantly being collected and analysed, leading to innovation and economic growth. Companies and organisations use the data they collect to personalise services, optimise the corporate decision-making process, predict future trends and much more. Today it is said that data is the new gold [2] [3]. As a result, data privacy has become one of the most important issues. While we all reap the benefits of a data-driven society, there is growing public concern about users' privacy. Centralised organisations accumulate vast amounts of personal and sensitive information. Individuals have no control over what data is stored about them and how it is used. In recent years, controversial incidents related to privacy have been repeatedly covered. There have been several attempts to address these privacy issues, both from a legislative perspective and from a technological point of view.

From legislative perspective, the General Data Protection Regulation (GDPR), the Health Insurance Portability and Accountability Act (HIPPA), the California Consumer Privacy Act (CCPA), the South Korea Personal Information Protection Act (PIPA), the National Strategy for Trusting Identities in Cyberspace (NSTIC), among others, propose reform of data protection and privacy rules, recommendations and regulations, recognising the growing need for personal data to be under the control of the individual (or organisation) to better mitigate the associated risks and address the difficult problem.

From a technological perspective, there exist plenty of security strategies and encryption algorithms which try to ensure that sensitive data would not be compromised. In addition, among them, most of the security strategies assume that only those who have secret keys can access the confidential data. However, the recent increase in reported incidents of surveillance

and security breaches compromising users' privacy call into question the current model, in which third-parties collect and control massive amounts of personal data. But there is an encryption scheme which does not require the use of the private key to compute with the encrypted data other than to decrypt the final result, the homomorphic encryption (HE) [4].

There are also new technologies, such as blockchain, that mitigate the need for trusted third parties. Bitcoin has demonstrated in the financial space that reliable and auditable computing is possible using a decentralised peer-to-peer network accompanied by a public ledger. Since the birth of Bitcoin, different approaches have been taken to the use of blockchain technology. The main and most popular application has been cryptocurrency, however, its desirable characteristics (immutability, security, verifiability, transparency) together with the invention of smart contracts have led to its successful application in many different fields, such as healthcare, insurance companies, logistics, digital identity... Despite the many advantages it offers, it is a technology that is not yet mature, and many challenges remain, being privacy one of them. The fact that its first application was cryptocurrencies has helped to spread the technology, which is currently booming. However, it needs to be developed further. For this reason, works of this kind and training in possible applications based on the use of blockchain play an important role in the development of this technology.

In short, over years the idea of strengthening privacy and the need for new technologies to protect has been growing. This way of thinking has changed many aspects of everyday life and many decentralised services where privacy is not respected have been challenged. Blockchain has shown that reliable and auditable computing is possible using a decentralised peer-to-peer network accompanied by a public ledger. Furthermore, homomorphic encryption guarantees confidentiality not only in the computation but also in the transmission and storage processes. The link between Blockchain and HE is growing rapidly in the computing environment. Over time, different projects have been designed in an attempt to improve privacy and return the power of data to the users themselves. Among these projects, it is worth highlighting the iDASH competition that promotes research into both technologies and the creation of new tools to improve the clinical environment [6].

1.2 General idea of our dapp

Our theoretical contribution is a decentralised application which enables computation on third party data without the need to reveal the data or the computation algorithm at any time. We propose a scenario where a data owner's data is uploaded to a distributed repository encrypted under their public key, so that only the data owner can decrypt it. An algorithm

owner then can take from the repository the encrypted data and use it for their algorithm. The main benefit here is to allow only the algorithm owner to have the results based on data from various sources without having to disclose this data to the algorithm owner, nor the algorithm and the result obtained from it to the data owners. This way, the algorithm owner does not learn anything about the data owner's data, except the result of the computation, and the data owner does not learn anything about the algorithm used or the algorithm's output.

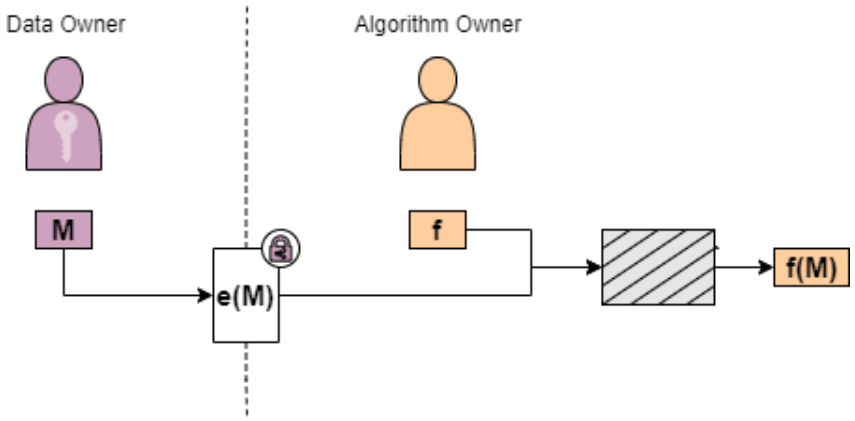

Figure 1.1: General idea of the protocol. M is the data, $\mathsf{e}(M)$ indicates the encryption of the data which can only be decrypted by the owner of the data, since the owner of the data is the only one who has the key to open the lock. The dashed line, like the colours, differentiates what each participant knows, being $\mathsf{e}(M)$ known by both. By means of the grey box with dashes we reflect a blackbox with inputs f and $\mathsf{e}(M)$ and output $\mathsf{f}(M)$. Which, being on the right side of the dashed line, is only known by the owner of the algorithm.

To put this in context, let us take a use case where the personal data of many people is needed, such as a clinical study for some research. Suppose researchers want to derive statistics from sensitive patient data but the function that they want to apply to the data will be known only to them, the researchers want to keep it secret. For a successful study it is necessary to have a large amount of data, but it is not always easy to find data providers and maintain a successful customer-provider relationship. Moreover, due to different data protection laws, data sharing can be a problem. Today such research is done with the data in the clear and, since the implementation of different laws, sensitive data should be private, even though they are often known to those who collect this data. To maintain privacy, we propose a system in which any researcher or organisation that has an algorithm and wants data to compute on it can access a decentralised database in which

they will find the encrypted data of different users, they can take those that meet the requirements they are looking for, operate on them and only get the final result after performing operations. Furthermore, in this model, in order to reward the data owner for their data, the algorithm owner gives some tokens to the data owner. We propose a protocol in which the owner of the algorithm and the data providers communicate with each other, without any third party, within an incentive system and all data is encrypted until the end.

Our idea is to make a Dapp that is accessible to everyone who wants to participate, both by sharing their data and by wanting to operate with third party data. We also want the interaction to be peer-to-peer, so that there is no need for third party intermediaries and that there is a record of what each participant does. To carry out theoretically our application, we will combine blockchain technology together with a Fully Homomorphic Encryption scheme. We propose a protocol that turns a blockchain into an application that allows interaction between the data owner and the algorithm owner that does not require trust between them neither a trusted third party, whit homomorphic encryption allows us to operate securely with encrypted data.

1.2.1 Problems we face in design

In the design of this scheme we have encountered two problems

1. **Problem: The result of the computation only has to be known by the owner of the algorithm**. If the key to decrypt the data is held only by the owner of the data, how is it possible that the result of operating f on the data is known only to the owner of the algorithm?

2. **Problem: Verifying that both parties play their part correctly**. In the application we want to design we have two parties, the owner of the data and the owner of the algorithm, who each have their private information (the data and the function, respectively) that they do not want or cannot publish. So, how can it be verified that the process has been carried out correctly and that there has been no corrupt participant?

We will answer these questions in Section 4.

Chapter 2

Blockchain

Lack of trust has repeatedly been identified as one of the most formidable barriers for people to engage in online commerce, involving transactions in which sensitive information such as personal, financial, healthcare information is sent to merchants over the Internet. History has already shown scandals such as the fraudulent use of data by Facebook or large websites hacked with data of millions of users leaked [7] [8] [9]. One potential approach to overcome these challenges is blockchain technology, which is currently popular in cryptocurrencies due to its properties of security, immutability and decentralisation. Until now there was no alternative but to trust *third trusted parties*, but now, with the advent of blockchain technology this situation reveals a need for change.

Blockchain is a Distributed Ledger Technology (DLT) that allows the recording of transactions between parties in a secure and permanent way, it can be understood as a decentralised, consensual and distributed database. But it goes further. This innovative technology aims to revolutionise online trust management, replacing third trusted parties with nodes. The nodes will be in charge of executing the code that will verify that the transaction is correct. Once the transaction is validated it is added to the blockchain ensuring that it is true, and once added it cannot be modified (if certain requirements are met). The big difference with the current system is that these nodes do not have access to falsify, modify or misuse the data of a transaction signed by a private key and since these keys are only owned by the user, they cannot be stolen in a hacker attack on a server. This is a breakthrough because in current systems, users' private keys are stored on servers that can be hacked, whereas in the case of Blockchain technology, if the administrator's private key is disclosed, the rest of the private keys remain secure because they are not stored on a compromised server.

2.1 History

Distributed Ledger Technology (DLT) refers to the ability of users to store and access information or records related to assets and holdings in a shared database, the ledger, capable of operating without a central validation system and based on their own standards and processes. DLTs differ from standard ledgers in that they are maintained by a distributed network of participants, called nodes, rather than a centralised entity.

It is difficult to mark the beginning of distributed ledgers. We could say that the history of ledgers goes back as far as 1494 where the first double-entry bookkeeping system was created, codified in a mathematical book in Italy [10]. But if we focus on the idea of distributed, we have to mark the 1990s as a key point. In these years a group of people started meeting to discuss programming and cryptography problems, which evolved into a mailing list discussing mathematics, computing, privacy, freedom, philosophy and politics. The group grew and ended up calling themselves "Cypherpunks". Specifically, in 1993 a movement appeared following the "Cypherpunk manifesto" whose main key was privacy. Based on this principle, the first ideas about digital currencies were developed [11] [12].

There were several attempts to implement anonymous transactions and privacy enhancements on the network. In 1991, the first work on a cryptographically secured blockchain was written by Stuart Haber and W. Scott Stornetta. They wanted to implement a system where the timestamps of documents could not be altered [13]. In 1992, together with Bayer, they incorporated Merkle trees into the design, which improved its efficiency by allowing multiple document certificates to be gathered into a single block [14]. After that, the first digital crypto-assets started to be created. In 1997 Adam Back created Hashcash. The idea was to impose a non-monetary cost on the sending of each mail to curb mass spamming. Other digital currency solutions followed, such as B-Money in 1998 by Wei Dai and Bit Gold by Nick Szabo in 2005.

The growing need for privacy and new technologies gave rise to what can be considered the first blockchain. On 31 October 2008, the concept of blockchain first emerged through a white paper, written by a person or group of people under the pseudonym Satoshi Nakamoto [15], which is currently known for being the base platform for the bitcoin cryptocurrency. It was developed to make bitcoin transactions on a platform where online payments could be sent directly from one peer to another without going through a financial institution. The main idea behind this project was to develop a trustless system that solves the double-spending problem using a peer-to-peer distributed ledger technology by computationally checking the chronological order of transactions. The blockchain owes its name to the way it works and the way it stores data. The information is packaged into blocks, which are linked together to form a chain. Each block plays a key role in

connecting to the previous block and the next. The main function of each block is to record, validate and distribute transactions between other blocks. This act of linking blocks into a chain is what makes the information stored on a blockchain so reliable. Once data is recorded in a block, it cannot be altered without having to change all the blocks that follow it, so it is impossible to do so without being seen by other participants in the network.

By 2010, the first Bitcoin transactions were made, which increased exponentially until nowadays. Subsequently, as Bitcoin's adoption and trust increased, new cryptocurrencies emerged, such as Ethereum, Bitcoin Cash, Ripple, EOS,...

2.1.1 Characteristics and types of blockchains

We can conclude by saying that blockchain is a decentralised information system that contains information about all past transactions and operates with a selected protocol [16] that defines the flow of transaction execution and validation, as well as the operation of the entire network and its members [17] [18] [19].

Hash functions are mathematical functions that convert digital information into fixed-size values. If that information is edited in any way, the hash also changes [20]. In blockchain networks, various transactions are grouped together to make a single block, which is then hashed. The blocks are connected to each other using the hashes of the blocks, each block contains its own hash along with the hash of the previous block.

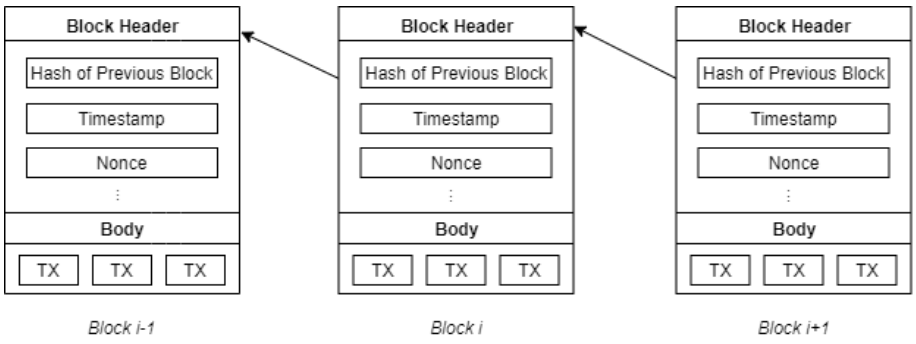

Figure 2.1: Example of a sequence of blocks

After a block has been added to the end of the blockchain, it is very difficult to go back and alter the contents of the block unless the majority reaches a consensus to do so. If an attacker tries to change any of the keys, the local registry will immediately become invalid, as the hash values of the headers of the following blocks will be completely different depending on the mechanism of the hash function. Therefore, the hash values are used to

validate the block's integrity. The property of immutability applies as the basic security feature of blockchain networks, the older the block is in the chain, the more the data in it is protected against changes.

Although when people think of the blockchain they often mistakenly refer to Bitcoin, where anyone can participate in the network, over time the needs have grown and new blockchains have been created. Depending on their characteristics, a classification can be made according to the access to data to control the execution of transactions [22]. In a *public* or *permissionless blockchain*, the blockchain is freely accessible to any user in the world who wishes to participate in it, in accordance with the consensus mechanisms of the community. This type of chain is characterised by the need for native tokens in order to incentivise users and maintain the system. The famous Bitcoin or Ethereum networks stand out. A blockchain network that operates in a restrictive environment like a closed network, or that is under the control of a single entity, is a *private blockchain*. Although it functions like a public blockchain network in the sense that it uses peer-to-peer connections and decentralisation, the access is only for those who have required prior authorisation and have been admitted. Private blockchains typically operate in a small network within a company or organisation. They are also known as permissioned blockchains or enterprise blockchains. A *consortium blockchains* is said to be semi-decentralised. Access is also authorised, but instead of a single organisation controlling it, several companies can operate a node in such a network. Finall, *hybrid blockchains* are a combination of public and private blockchains. In a hybrid blockchain the participating nodes are invited, but all transactions are public. This means that nodes participate in the maintenance and security of this blockchain, but all transactions are visible to users worldwide, unlike private blockchains in which transactions are private as well.

2.2 Ethereum and Smart Contracts

As we have seen, depending on the need, we will have to opt for one type of blockchain or another. Specifically, as our idea for the project is to be public and accessible to all, we will opt for Ethereum, as in addition to being public, it has certain interesting features for the project, as we will see below.

Ethereum [24], introduced by Vitalik Buterin, is a smart contract network that allows users to make simple currency transfers in its native currency, Ether, as well as to deploy complex applications by creating user-defined smart contracts. To understand roughly what Ethereum means, it can be said that Ethereum is a large computational system, which can be freely used, or even modified, to meet any need. Moreover, one can create its own coins and offer them to whoever is willing to consume them.

Ethereum aims to build a system where anyone can create a decentralised system that works with blockchain technology. To do this, Ethereum introduces a Turing-complete language [25] and maintains a virtual machine to execute contracts written in this language, called smart contracts.

The origin of smart contracts dates back to 1997, the year in which Nick Szabo published the article *Formalizing and Securing Relationships on Public Networks* [26] in which the concept of smart contract is presented as a transaction protocol that theoretically executes the terms of a contract. It was not until 2004, when Vitalik Buterin and Gavin Wood created Ethereum, bringing together the concepts of smart contract and blockchain in a single platform and made it real [27]. A smart contract is a programm stored on the blockchain, bearing some analogy to traditional contracts. While traditional contracts are subject to laws, they are self-executing and self-enforcing computer contracts, enforcing the agreement between the parties, without the need for mediators. Thanks to the immutability and transparency properties of their technology, smart contracts are secure and reliable, with no risk of manipulation and forgery. The use of such contracts allows developers to implement what are known as decentralised applications (dapps), where it is possible to define one's own rules of possession and transaction formats that are fully customised for the specific application domain. When creating a smart contract, the developer defines its behaviour and the terms of the agreement directly in lines of code. Once the contract is included in the blockchain it cannot be modified, thus ensuring that the execution of the contract will be carried out as programmed. Smart contracts can be seen as independent programs stored on the blockchain that will autonomously carry out transactions, operations or actions if certain conditions are met. These actions and conditions are specified by the developer depending on the scope and purpose of the application being implemented.

Ethereum is based on an account-based model instead of the UTXO model like Bitcoin [23]. Therefore, it introduces a more advanced notion of the state of the ledger, which includes the state of all accounts in the system. The blockchain is equivalent to a transaction-based state machine [24]. A state is defined by objects called accounts. In turn, each account is associated with a 20-byte address. Transitions between states are represented by the transfer of value and information between accounts, that is, by the execution of transactions. The information necessary to carry out the validation of the terms collected in each transaction is available in the previous state.

Ethereum offers two types of accounts: externally owned accounts (EOA) that are controlled by users and contract accounts that are controlled by their contract code. The state of an EOA consists mainly of a nonce (to prevent replay attacks) and a balance, while state of a contract account also includes the contract code and its storage. Both types of accounts can

invoke functions of a smart contract code, however, only an EOA can initiate a transaction or deploy a smart contract. Miners will execute the code of any smart contract when invoked. To avoid denial-of-service (DoS) attacks, each transaction on Ethereum has an associated cost in terms of gas. This gas is used to limit the total computational capacity associated with the transaction. This cost is necessary as a mechanism to control the amount of resources allocated to the execution of a transaction in the Ethereum Virtual Machine (EVM) [28]. The EVM can handle code of any complexity and can be programmed in several existing programming languages, such as Solidity or Serpent, and this is what allows users to build their own operations and applications. The Ethereum virtual machine is considered the heart of Ethereum, as it is the hub of the platform. It is the system that allows all the nodes to be connected and acts as the engine of the platform, executing smart contracts [29]. At all times, the Ethereum ledger is updated by hundreds of nodes connected in the same network, which are running the EVM and the same instructions. Therefore, we can say that Ethereum could be a big world computer, where each node executes the EVM in order to maintain the same consensus on the blockchain.

We can conclude by saying that Ethereum is a programmable blockchain, differing from other blockchains in that, instead of offering a set of predefined transactions, Ethereum allows users to build their own transactions and applications. Ethereum stands out for its simplicity and universality. It provides a high-level internal language, which any programmer can use to build any smart contract or invent their own currency. Ethereum is designed to be as modular as possible, that is to say that during development, the main goal is that if a modification is made to the protocol, the stack will still work. In addition, the Ethereum protocol does not restrict specific features of use. For this reason, and because of the basic characteristics that blockchain offers, we have chosen Ethereum as the tool for our project.

Chapter 3

Homomorphic Encryption

Homomorphic encryption is a fairly new technology that allows, as a cipher scheme, the encryption and decryption of messages. But it goes further, as it allows operations to be performed on the ciphertexts, resulting in a new ciphertext. In this chapter we provide a background of the history of homomorphic encryption and we formalize what it is following [30], [31], in order to better understand the next chapter. We are not going to explain it in detail since it is not necessary to follow our proposal and it is not the goal of this work. The main purpose of this chapter is to explain only the necessary notions to understand our protocol.

As already mentioned, homomorphic encryption is characterised by enabling operations on encrypted messages based only on publicly available information, and in particular without having access to any secret key. More specifically, it allows certain computable functions to be performed on ciphertexts using only publicly available information without any risk of data breach. To understand this better we will explain one of the first homomorphic encryption system, the RSA cipher [32]. It simply uses the concept of modular exponentiation which says that the modulus N is the product of two large primes p and q and the public key and private key are e and d respectively that meet $ed \equiv 1(\mod \phi(N))$. The encryption function e is performed using the public key e as $\mathsf{e}(m) = m^e(\mod N) = c$, where m is the plaintext such that $0 < m < N$. The ciphertext c can be decrypted using the decryption function d and the private key d, $\mathsf{d}(c) = c^d(\mod N) = m$. The RSA cryptosystem has a multiplicative homomorphic property, i.e., the encryption of the product of the plaintexts can be obtained from the ciphers of these plaitexts. Given two plaintext m_1 and m_2, the result of the multiplication of the ciphertexts will be the ciphertext of the product:

$$\mathsf{e}(m_1)*\mathsf{e}(m_2) = {m_1}^e \pmod N * {m_2}^e \pmod N = (m_1 * m_2)^e \pmod N = \mathsf{e}(m_1 * m_2).$$

This shows that it is possible to obtain the cipher of the multiplication of the clear plaintexts, without knowing them, using only their ciphers.

It is important to note that in order to obtain the same result as a normal addition or multiplication between plaintexts, very often slightly different operations are performed in the ciphertext space. But given the encrypted data and the equivalent operations, it is possible to calculate a cipher that when decrypted gives the same result. In this RSA example we have just seen, the multiplication used between the two plaintexts can be applied directly to the respective ciphertexts, although this is not normally the case. For example, in a well-known cipher such as the Paillier cryptosystem [33], it can be seen that in order to calculate the cipher of the sum of two plaintexts, it is necessary to multiply the corresponding ciphertexts, as we will see in the following without going into details. The encryption of a plaintext m is $\mathsf{e}(m) = g^m r^N (\mod N^2)$, where N and g are the public key and r is a random number, so when two ciphertexts are multiplied, the result is a equivalent [I] encryption to the sum of their plaintexts:

$$\begin{aligned} \mathsf{e}(m_1) * \mathsf{e}(m_2) &= {g^{m_1}{r_1}^N} * {g^{m_2}{r_2}^N} (\mod N^2) = g^{m_1+m_2}(r_1 r_2)^N (\mod N^2) \\ &= g^{m_1+m_2} r^N (\mod N^2) \equiv \mathsf{e}(m_1 + m_2) \end{aligned}$$

In other words, the result of the multiplication of two ciphertexts decrypts into the sum of their plaintexts, $\mathsf{d}(\mathsf{e}(m_1) * \mathsf{e}(m_2)) = m_1 + m_2$.

In general, we will have to find a function applicable to the ciphertext that is equivalent to the function we want to apply to the plaintext, which we will call a homomorphic function and we will indicate as f_h. A homomorphic function applied to ciphertexts gives the same result, after decryption, as applying the function to the original unencrypted data. In a Fully Homomorphic Encryption, given two ciphertext of x and y as e_x and e_y, it should be possible to compute $e_{\mathsf{f}(x,y)}$ for any function f, without having access to any secret information. This would require finding the equivalent homomorphic function f_h to apply to e_x and e_y. We emphasize that normally f and f_h are not the same function. In particular, two ciphertexts of x and y can be added/multiplied together homomorphically and the result will be a new ciphertext that encrypts the sum/multiplication of x and y. We will use the notation $+_h$ and $*_h$ to indicate that operations performed between the ciphertexts are, respectively, the equivalent of addition and multiplication between the plaintexts. In addition, to indicate that two ciphers are equivalent, that is, that they give the same result when decrypted, instead of using $=$ we use the symbol $\equiv$. In the drawings we will use squares to indicate that the message inside is encrypted, as can be seen in the drawing below.

[I] It is equivalent and not equal because of the randomness of r

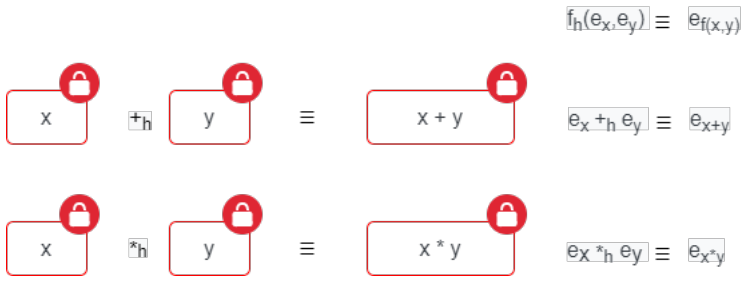

Figure 3.1: Basic operations

3.1 History

Homomorphic encryption was first considered in 1978, when three researchers named Rivest, Adleman and Dertouzos [34] started talking about privacy homomorphism, as they called it at first. In the RSA case, the encryption and decryption functions e and d are homomorphic algebraic functions. And this is the reason why this type of encryption is called now *homomorphic*. Actually, the RSA public encryption scheme came out at that time and was the first example of a Partially Homomorphic Encryption (PHE) scheme, as it is only multiplicatively homomorphic. This was a great idea, but it was not clear whether a Fully Homomorphic Encryption (FHE) was achievable. These terms will be introduced in more detail later on and the distinction will become clear. For now, it suffices to know that a FHE can compute anything on encrypted data, while the others are more restricted. Since that proposal, for more than 30 years, researchers have been confronted with the problem that the number of operations that can be performed is limited. To understand this better we have to introduce the concept of noise. The noise is usually a small term that is added to the ciphertext while it is being encrypted. This noise is added to ensure the security[II] of cryptosystems and is drawn from a distribution of small terms. Therefore, the noise is random. The decryption function does not work if the noise is greater than a certain maximum value (each scheme has its own set of parameters and therefore has its own limit). Homomorphic operations increase the noise and therefore the number of operations is usually limited, if the noise reaches the threshold, then the decryption is incorrect.

We will explain this better by means of Figure 3.2 and an example of a simple symmetric cryptosystem that can be found in [35]. This cryptosystem encrypts a bit m into an integer $c = pq + 2r + m$ where p is the secret key and q and r are random positive integers. To decrypt c, it has to be

[II]The term noise denotes a moderate amount of error injected into the encrypted message and generates a non-exact relationship, thus brute-force attacks are prevented.

reduced first modulo p, $c \mod p$, and then modulo 2, $\mod 2$, and only if the term $2r+m$ is less than p, the result will be m, otherwise the decryption function fails. In this cryptosystem $2r$ is the noise and the threshold is $p-m$. This cryptosystem theoretically allows to perform homomorphically both addition and multiplication. Given two ciphertexts $c_0 = pq_0+2r_0+m_0$ and $c_1 = pq_1 + 2r_1 + m_1$:

$$c_o + c_1 = p(q_0 + q_1) + 2(r_0 + r_1) + m_0 + m_1$$

$$c_o * c_1 = p(q_0 pq_1 + q_0(2r_1 + m_1) + q_1(2r_0 + m_0)) + 2(2r_0 r_1 + r_0 m_1 + r_1 m_0) + m_0 m_1$$

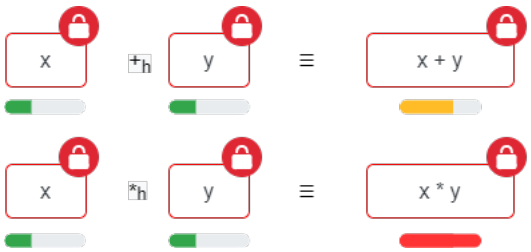

Figure 3.2: The noise grows. We have drawn the case of the multiplication that passes the threshold, by painting it in red and colouring up to the top. The result of the homomorphic addition, on the other hand, is coloured in yellow and not to the top, indicating that it can still be deciphered correctly but it is quite close to reaching the threshold.

We can clearly see that addition increases the noise. Multiplication increases it even more, and this is not a coincidence. In most of the homomorphic schemes known so far, the way noise grows with multiplication is more severe than in the case of addition. For this reason, the most important metric for quantifying the hardness of homomorphically evaluating a circuit is its multiplicative depth.

We have already used the terms SHE, PHE and FHE, but what are they really? How do they differ? We will first explain what these concepts are without getting into formalization. *Partially Homomorphic Encryption Schemes, PHE*, are schemes that support either addition or multiplication, but not both. The RSA cryptosystem is the first multiplicative PHE, ElGamal [36] is another relevant multiplicative PHE. Shafi Goldwasser and Silvio Micali's cryptosystem [37] is the first additive PHE. *Somewhat homomorphic encryption schemes, SHE*, are those schemes that can evaluate both homomorphic operations on encrypted inputs, but for which proceeding further would result in the loss of decryption correctness. The scheme of Dan Boneh, Eu-Jin Goh and Kobbi Nissim [38] was the first approach that allowed both addition and multiplication with constant-size ciphertexts. The

last example of the cryptosystem we have used is also SHE. Each encryption scheme is set up with some parameters, e.g. the size of the primes that are used, the size of the secret key, etc. Note that the following notion is stronger than the Somewhat Homomorphic Encryption scheme. A homomorphic encryption scheme is said to be *Leveled* if, for any multiplicative depth L fixed a priori, it is possible to find a set of parameters such that the encryption scheme instantiated with those parameters is able to homomorphically evaluate any circuit of depth L. At some point, in any of the above three types of schemes, the noise level could be too large and it would be impossible to proceed without losing the correction. We are interested in those schemes that can homomorphically evaluate an unlimited number of operations. Such a scheme is called *Fully Homomorphic Scheme, FHE.*

During more than 30 years there has been a desire to achieve this purpose of having a way to compose the two fundamental operations together as having a scheme that supports both operations homomorphically. It wasn't until 2009 when Gentry, an IBM student at the time, developed a Fully Homomorphic encryption solution called bootstrapping. Gentry published the first example of a scheme that supports both the additional and multiplication operations [39]. Since then, lattices became even more popular among cryptography researchers and a new era in cryptography was born. Research began to produce a number of ambitious cryptosystems, each with its own features and improvements over the previous ones. The advance in FHE can be grouped into four main families:

1 *Ideal Lattice-based* schemes. They follow Gentry's original idea, whose hardness security is based on the lattice reduction problem.

2 *Over Integers* schemes. This family includes works that the hardness of the schemes is based on the Approximate of Greatest Common Divisor (AGCD) problem. In 2010 the DGHV (Dijk, Gentry, Halevi and Vainkuntanathan) [35] scheme was published, using regular or modular integers, which is a basic arithmetic that allows understanding but is not efficient and therefore it is not used in current applications.

3 *(Ring) Learning With Error* schemes. This family includes schemes based on Learning with Error (LWE) and Ring Learning with Error (RLWE), where both approaches are reducible to the lattice problems. In 2011 the first Fully Homomorphic scheme based on LWE problems called BGV (Brakerski and Vaikuntanathan) [40] and in 2013 the GSW scheme (Gentry, Sahai and Waters) [41] were released.

4 *Nth-Degree Truncated Polynomial Ring Unit (NTRU)* schemes. This family is also based on the lattice problem. NTRU encryption scheme is old and standardized lattice-based encryption scheme homomorphic properties were realized lately. In 2012 LTV (López-Alt-Tromer-Vaikuntanathan) was released [42], but it has some security problems.

The continuous improvements and new approaches developed are gradually increasing the efficiency and performance of FHE schemes. Nowadays, most of the studies are based on LWE. In 2016, the CKKS (Cheon, Kim, Kim and Song) [43] and in 2019 the Torus Fully homomorphic Encryption, TFHE [44], schemes were published. One of the latest projects that uses TFHE and is gaining attention is ZAMA in which major cryptographers such as Paillier are involved [5]. The timeline of the relevant FHE approaches has been summarised in the following figure.

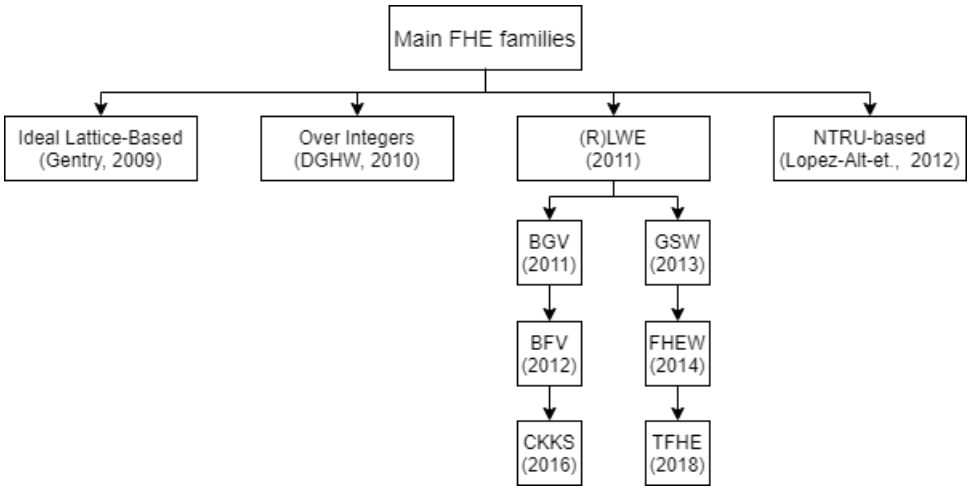

Figure 3.3: Main FHE families

3.2 Formalizing FHE

FHE is intended to produce the image of a ciphertext $\mathsf{f}(e_m)$ for any function f and any ciphertext message e_m of a plaintext m, requiring that no information about $\mathsf{f}(m)$ or m is leaked. The operation of an FHE scheme can be seen as the classical black box model in computer systems.

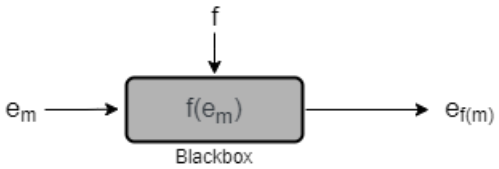

Figure 3.4: Homomorphic Evaluate ϵ

The general idea behind FHE is that the function f can be efficiently expressed as a circuit that processes homomorphically encrypted data. In

most FHE schemes circuits are evaluated. Although the idea of viewing circuits in terms of gates AND-, OR- and NON-gates is widespread, it turns out that circuits can be built from other types of gates that are themselves functionally complete gates [45]. One of these gates is the NAND gate, and, as we will see, if a scheme can compute at least one NAND gate (in addition to its own decryption circuit, as we will see) then it can compute any circuit.

It is important to note that the inputs and the output of the circuit are ciphertexts. After evaluating too many gates, the noise becomes too big and the output can no longer be correctly decrypted. What Gentry came up with was a way to reduce the noise and pass from a Somewhat Homomorphic Encryption scheme to a fully one, such technique was called bootstraping. Put it in a non-technical way, what Gentry called bootstrapping was the process of refreshing a ciphertext in order to produce a new cipher that encrypts the same message, but with a lower level of noise so that more homomorphic operations can be evaluated on it. This operation consists of homomorphically evaluating the decryption circuit of the scheme.

Before formalising theoretically how bootstrapping works, which is the key point to understand our project, we need some basic notions regarding the evaluation of circuits.

Definition 3.2.1. Let $\mathcal{C}$ be a set of circuits. A $\mathcal{C}$ *evaluation escheme* is composed by four operation algorithms: KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$, and $\mathsf{Evaluate}$. The computational complexity of all operations are set with respect to a security parameter λ, where:

- $\mathsf{KeyGen}(\lambda) \rightarrow (pk, sk, ek)$: given the security parameter λ outputs a public key pk used for encryption, private key sk used for decryption and evaluation key ek used for evaluation.

- $\mathsf{Encrypt}(m,\ pk) \rightarrow c$: given the public key pk and a message m, it outputs a ciphertext c.

- $\mathsf{Decrypt}(c,\ sk) \rightarrow m$: given the secret key sk and a ciphertext or an evalutaion output c, it outputs a message m.

- $\mathsf{Evaluate}\ (c_1, ..., c_l,\ ek,\ C) \rightarrow e$: given the evaluation key ek, a circuit $\mathsf{C} \in \mathcal{C}$, and a tuple of ciphertexts or/and evaluation output $\bar{\mathrm{c}} = (c_1, ..., c_l)$; it outputs an evaluation e.

It is standard to assume that the evaluation key and the public key are the same. The use of an evaluation key was first defined in [46]. By defining the scheme with an evaluation key separate from the public key, what is meant is that it is not necessary that $ek = pk$. In particular, the evaluation key can be part of the public key. In this work, as we are not going into detail about the $\mathsf{Evaluate}$ function, we will always assume that

$ek = pk$ and so, from now on, we can omit the evaluation key from the notation, understanding that calculating $\mathsf{Evaluate}(\mathsf{e}_{pk}(m), C)$ means calculating $\mathsf{Evaluate}(\mathsf{e}_{pk}(m), pk, C)$.

The following definitions are fundamental to any homomorphic encryption scheme and they can be expressed using the four basic operations defined above. They are correctness and compactness. Intuitively, the concept of a scheme being correct is that all four operations work well and so applying the decryption operation to an evaluated output yields what is expected. Formally,

Definition 3.2.2 (Correctness)**.** A $\mathcal{C}$ evaluation scheme is *correct* if, for any key-pair (pk, sk), any circuit $\mathsf{C} \in \mathcal{C}$ and any tuple that can be a mix of ciphertexts and evaluation outputs $\bar{\mathrm{c}} = (c_1, ..., c_l)$ for plaintexts $\bar{\mathrm{m}} = (m_1, ..., m_l)$, the following is met:

$$e = \mathsf{Evaluate}\ (\bar{\mathrm{c}},\ C),\ \text{then}\ \mathsf{Decrypt}(e,\ sk) = \mathsf{C}(\bar{\mathrm{m}})$$

Definition 3.2.3 (Compactness)**.** A $\mathcal{C}$ evaluation scheme is *compact* if there is a polynomial p, such that for any key-pair (sk, pk) output by $\mathsf{KeyGen}(\lambda)$, any circuit $\mathsf{C} \in \mathcal{C}$ and any tuple that can be a mix of ciphertexts and evaluation outputs $\bar{\mathrm{c}} = (c_1, ..., c_l)$, the size of the output $\mathsf{Evaluate}\ (\bar{\mathrm{c}},\ C)$ is not more than $\mathsf{p}(\lambda)$ bits, independent of the size of the circuit.

In other words, if a $\mathcal{C}$ evaluation scheme is compact, it means that the ciphertext size does not grow much through homomorphic operations and the output length only depends on the security parameter λ. Without compactness the homomorphic evaluation could increase the size of a ciphertext, which could lead to increased decryption time.

As mentioned before, we can use these two concepts to formalise the different types of homomorphic schemes. As our aim is to arrive at an FHE scheme, we will only formalise the idea of SHE, since this scheme allows us to operate both basic operations and is where Gentry started from to arrive at an FHE scheme. Let's go there.

Definition 3.2.4. Let $\mathcal{C}$ be a set of circuits. A $\mathcal{C}$ evaluation scheme (KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$, $\mathsf{Evaluate}$) that is correct is called *Somewhat Homomorphic Encryption* scheme.

Definition 3.2.5. Let $\mathcal{C}$ be the set of all circuits. A $\mathcal{C}$ evaluation scheme (KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$, $\mathsf{Evaluate}$) that is compact and correct is called *Fully Homomorphic Encryption* scheme.

The major difference between the two definitions is that to be fully requires that $\mathcal{C}$ is the set of all circuits and not just the set of some circuits as in the case of SHE. This is intuitive since a Fully Homomorphic Encryption can compute anything on the encrypted data and a Somewhat Homomorphic

Encryption is more restricted and can only compute a restricted amount of circuits. The other difference, which is not very important but is striking, is that in the SHE definition there is no compactness requirement. We could omit this requirement in the definition of FHE, but not taking compactness into account makes Fully Homomorphic Encryption trivial. Let's see why. To create an non-compact FHE it is sufficient to take any encryption scheme, add the identity function as Evaluate to it and modify the Decryption function so that when the scheme receives ciphertexts and a circuit as input, first the Evaluate function (identity) returns the ciphertexts and the circuit and then the Decryption function decrypts the ciphertexts and evaluates the circuit on the plaintexts [47], [48]. This delegates the actual computation to the Decryption algorithm. If we remove the compactness requirement from Definition 3.2.5, then this scheme would be FHE, but for obvious reasons we do not want a Fully Homomorphic Encryption scheme to work like this, so the compactness property is requested. Note that the more complex the circuit is to evaluate, the higher will be the output of the Evaluate function. This completely breaks the compactness condition. Moreover, in the most commonly used schemes, the spaces of plaintexts, ciphertexts, keys, etc. are usually finite spaces, so it is also natural to ask for the output of the Evaluate function to be controlled.

3.2.1 Bootstrapping

In this section, we will see a method to go from a SHE scheme to a FHE one. As we have been saying, in SHE schemes, after some operations, the ciphertext returned by the evaluation function has a high noise that must be reduced in order to continue operating. Decrypting the ciphertext reduces the noise, but decrypting it leaves the data in the clear, and that is precisely what we are trying to avoid with a homomorphic scheme. To avoid this, Gentry gave the first solution by performing a double cipher to achieve a fully HE scheme. Let f be the function that we want to operate on the plaintext m. As explained in Section 3.2, let's look at f as a circuit written with NAND gates and apply them one by one until the ciphertext gets a critical noise that is very close to the threshold. Let us call that cipher c and assume it is encrypted with the pk of the key pair (pk, sk). At this point, in order to decrypt c Gentry proposes to re-encrypt c with a new key pk' (we will call its pair sk'). This increases the noise, but now we can consider removing the first cipher, being left with only the noise of the second cipher. In this way we would reduce the noise, but not eliminate it. This is because c is a cipher that has a very high noise, when decrypting it the noise is eliminated but having also encrypted with the new key pk' the noise persists, but less high, since it is only the noise added by the encryption function. With this new cipher no operation has been performed yet, so the noise it has now is as low as possible.

To decrypt the first cipher Gentry proposes to send the decryption function as a circuit to the evaluate function. The key to decrypt in this case would be the encryption of the second secret key sk', using the first public key pk. In this way a new ciphertext is obtained from the same plaintext m but encrypted with another key pk' and with less noise. This idea is known as bootstrapping and to understand it better we have captured it in the following figure. In there you can see the bootstrapping steps, the functions to be applied in each step and what is obtained after each step. By means of the red and blue colours we have differentiated the key pairs of the first and second cipher: in red (pk, sk) and in blue (pk', sk'). We have used the coloured squares to indicate that what is inside is encrypted with the corresponding coloured key. We have also indicated the noise of the cipher for each step.

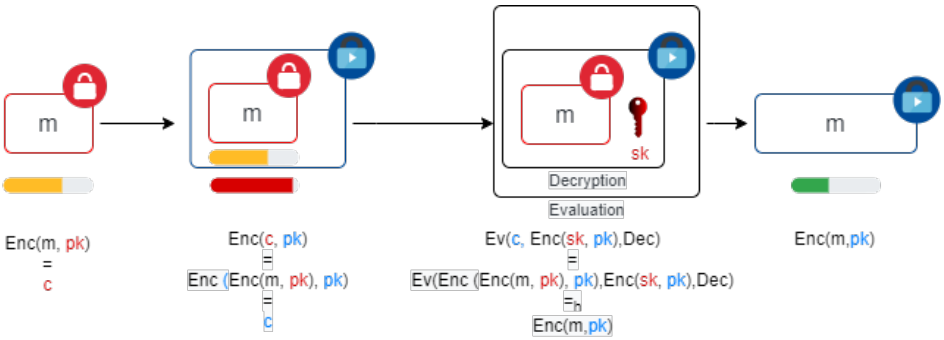

Figure 3.5: Gentry's bootstrapping idea.

That is, in order to turn a SHE scheme to a FHE scheme we need the evaluation function to be able to evaluate the decryption function. Moreover, since we want it to be able to continue operating, we need it to be able to operate one more logic gate. Then it will be able to evaluate the next logic gate of the circuit it is evaluating. Therefore, for the scheme to be fully functional it has to be able to evaluate the number of logic gates of the decryption function as a circuit plus one.

Definition 3.2.6. Let (KeyGen, Encrypt, Decrypt, Evaluate) be a $\mathcal{C}$ Somewhat Homomorphic Scheme. Assume C contains the operation Decrypt (expressed as a circuit of NAND gates) followed by one extra NAND gate. Then the scheme is said to be *bootstrappable*.

We now have all the pieces to understand Gentry's idea of moving from a SHE scheme to a FHE scheme. Suppose we have a SHE scheme that is able to homomorphically evaluate its decryption function and an extra NAND gate, i.e., it is bootstrappable. Let's call it *Bootstrappable Somewhat Homomorphic*. We will continue to use the notation and assumptions made at the

beginning of this section. Thus, c is the encryption of m after having applied several logic gates and which has a very high noise. To start with the bootstrapping method, choose a new public key pk' (with sk' its corresponding secret key) and $\widetilde{sk} = \mathsf{Encrypt}(sk, pk')$. Re-encrypt c with the new public key pk', and let c'=$\mathsf{Encrypt}(c, pk')$. Now decrypt the double encryption c' using the key $\widetilde{sk}$:

$$\begin{aligned} \mathsf{Dec}(c', \widetilde{sk}) &= \mathsf{Dec}(\mathsf{Enc}(c, pk'), \mathsf{Enc}(sk, pk')) \equiv \mathsf{Enc}(\mathsf{Dec}(c, ks), pk') \\ &= \mathsf{Enc}(\mathsf{Dec}(\mathsf{Enc}(m, pk), sk), pk') = \mathsf{Enc}(m, pk') \end{aligned}$$

The result obtained has a lower noise level, so it allows us to continue operating the remaining gates of the circuit we are evaluating until we obtain another cipher with a high noise level. At that point we would have to do the bootstrap again. So, this step has to be taken every time that after operating several logic gates of the circuit the obtained ciphertext has a high noise. This means that it is a recursive algorithm and therefore we can summarised it as follows. Given any bootstrappable SHE scheme, the evaluation process is the following:

Algorithm 1 From botstrappable SHE to FHE

Input. $\mathcal{C}$ Botstrappable SHE scheme, C circuit written with NAND gates.
Output: Evaluation of a set of plaintext M using C.
Start
    `While` there is any gate from C to analyse:
        `Evaluate` the next C's NAND gate.
        `If` the noise is reaching the threshold:
            `Bootstrapp`.
fin

As mentioned before, this was the first proposal to achieve a Fully Homomorphic cipher, but not the only one, as over the years new techniques have been devised. But we will not go any further into this world, as the process seen in this section is sufficient to understand our proposal written in the next chapter.

Chapter 4

Dealing with the theoretical issues

In this chapter we will discus the two main theoretical problems we have seen in designing our protocol and we propose for each problem a possible solution. Finally, we have tried to combine both solutions in order to cover both problems with a single solution.

4.1 First problem: How to proceed so that only the owner of the algorithm gets the clear result?

As mentioned in Chapter 1, the first problem we faced when designing the flow of the protocol is how to prevent anyone from getting the unencrypted result of the computation other than the owner of the algorithm, without changing the logic of the service. Once the algorithm owner has performed the corresponding computations on the encrypted data, we must introduce a way by which they can obtain the decryption of the result of the computation, but this must be done so that only they, the algorithm owner, obtains such decryption. As it is, the private key at this point is held by the data onwer. If the algorithm owner sends the result of the computation to the data owner so that they can decrypt it, we will be missing our purpose which is that only the algorithm owner obtains the result of applying the algorithm on the unencrypted data. To solve this issue we propose the following.

This idea is taken from the bootstrapping method. As we have explained in Section 3.2.1, in order to continue operating on encrypted data, bootstrapping allows us to reduce the accumulated noise by switching keys. A natural solution to the problem we discussed above is that once the algorithm owner gets a result after operating the encrypted data on their algorithm, instead of sending it to the data owner, they do an extra step, similar to what they

were doing in the bootstrapping method but with a difference: instead of using the auxiliary keys provided by the data owner, using a new key pair (pk^A, sk^A) generated by them.

4.1.1 Solution to the first problem: Double encryption

At this point, we detail step by step our proposal for a better understanding and as a proof that it can be done. It should be noted that this solution is valid for any bootstrappable Somewhat Homomorphic Encryption scheme, so we will not specify any and use a general SHE system with encryption and decryption algorithms denoted by e and d, respectively.

Protocol 1. Double Encryption Scheme

Private inputs.

DataOwner. Sensitive data $M = \{m_1, ..., m_n\}$.
AlgorithmOwner. Algorithm f.

Goal. To compute the data set M in the algorithm f, using the corresponding ciphertexts provided by the owner of the data and making the result public only to the owner of the algorithm.

The protocol:

1. **DataOwner**
   (a) generates a pair of keys to encrypt and decrypt (pk, sk).
   (b) encrypts each value of the data set M with the pk key: $\mathsf{e}_{pk}(M)$.
   (c) generates a sequence of key pairs (pk_i, sk_i) to generate the auxiliary keys for the bootstrapping method $(pk_i, \widetilde{sk_i})$ where $\widetilde{sk_i} = \mathsf{e}_{pk_{i-1}}(sk_{i-1})$ for all $i \in \{1, ..., t\}$ and $\widetilde{sk_1} = \mathsf{e}_{pk}(sk)$.
   (d) makes public the encrypted data $\mathsf{e}_{pk}(M)$ and the auxiliary keys for bootstrapping $\{(pk_i, \widetilde{sk_i}) | i \in \{1, \cdots, t\}\}$.
2. **AlgorithmOwner**
   (a) decomposes f into circuits $(f_1, ..., f_l)$ such that each f_i consists of a number of NAND gates that does not exceed the maximum noise permited to perfom decryption correctly, $\mathsf{f} = f_l \circ \cdots \circ f_1$.
   (b) calculates $\mathsf{f}(\mathsf{e}_{pk}(M))$ using the bootstrapping method, and gets as a result $\mathsf{e}_{pk_l}(\mathsf{f}(\mathsf{M}))$. The way this is done is the following:
      b1. sends encrypted data $\mathsf{e}_{pk}(M)$ and the circuit f_1 to the evaluation function, $\mathsf{evaluate}(\mathsf{e}_{pk}(M), \mathsf{f}_1)$, and obtains $r_1 = f_1(\mathsf{e}_{pk}(M))$.

As the noise is high, it has to be reduced. For this, initializes the bootstrapping method with the first pair of keys $(pk_1, \widetilde{sk_1})$ as follows:

1.1 calculates $r'_1 = \mathsf{e}_{pk_1}(r_1)$.

1.2 decrypts r'_1 using the $\widetilde{sk_1}$ key and as a result gets $\mathsf{e}_{pk_1}(f_1(M))$ that it has a reduced noise that allows it to continue to operate:

$$\begin{aligned}\mathsf{d}(r'_1, \widetilde{sk_1}) &= \mathsf{d}(\mathsf{e}_{pk_1}(r_1), \mathsf{e}_{pk_1}(sk)) \equiv \mathsf{e}_{pk_1}(\mathsf{d}(r_1, sk)) \\ &= \mathsf{e}_{pk_1}(\mathsf{d}(f_1(\mathsf{e}_{pk}(M)), sk)) \\ &\equiv \mathsf{e}_{pk_1}(\mathsf{d}(\mathsf{e}_{pk}(f_1(M)), sk)) = \mathsf{e}_{pk_1}(f_1(M)).\end{aligned}$$

...

bi. operates $\mathsf{evaluate}(\mathsf{e}_{pk_{i-1}}(\mathsf{f}_{i-1}(M)), \mathsf{f}_i)$ to obtain $r_i = f_i(\mathsf{e}_{pk_{i-1}}(M))$ and using the pair of keys $(pk_i, \widetilde{sk_i})$ does:

i.1 calculates $r'_i = \mathsf{e}_{pk_i}(r_i)$.

i.2 decrypts r'_i using the $\widetilde{sk_i}$ key and as a result gets $\mathsf{e}_{pk_i}(f_i(M))$:

$$\begin{aligned}\mathsf{d}(r'_i, \widetilde{sk_i}) &= \mathsf{d}(\mathsf{e}_{pk_i}(r_i), \mathsf{e}_{pk_i}(sk_{i-1})) \equiv \mathsf{e}_{pk_i}(\mathsf{d}(r_i, sk_{i-1})) \\ &= \mathsf{e}_{pk_i}(\mathsf{d}(f_i(\mathsf{e}_{pk_i}(M)), sk_{i-1})) \\ &\equiv \mathsf{e}_{pk_i}(\mathsf{d}(\mathsf{e}_{pk_i}(f_i(M)), sk_{i-1})) = \mathsf{e}_{pk_i}(f_i(M)).\end{aligned}$$

...

bl. operates $\mathsf{evaluate}(\mathsf{e}_{pk_{l-1}}(\mathsf{f}_{l-1}(M)), \mathsf{f}_l)$ to obtain $r_l = f_l(\mathsf{e}_{pk_{l-1}}(M))$ and using the pair of keys $(pk_l, \widetilde{sk_l})$ does:

l.1 calculates $r'_l = \mathsf{e}_{pk_l}(r_l)$.

l.2 decrypts r'_l using the $\widetilde{sk_l}$ key and as a result gets $\mathsf{e}_{pk_l}(f_l(M)) = \mathsf{e}_{pk_l}(\mathsf{f}(M))$:

$$\begin{aligned}\mathsf{d}(r'_l, \widetilde{sk_l}) &= \mathsf{d}(\mathsf{e}_{pk_l}(r_l), \mathsf{e}_{pk_l}(sk_{l-1})) \equiv \mathsf{e}_{pk_l}(\mathsf{d}(r_l, sk_{l-1})) \\ &= \mathsf{e}_{pk_l}(\mathsf{d}(f_l(\mathsf{e}_{pk_l}(M)), sk_{l-1})) \\ &\equiv \mathsf{e}_{pk_l}(\mathsf{d}(\mathsf{e}_{pk_l}(f_l(M)), sk_{l-1})) = \mathsf{e}_{pk_l}(f_l(M)) \\ &= \mathsf{e}_{pk_l}(\mathsf{f}(M)) = r.\end{aligned}$$

(c) generates a pair of keys to encrypt and decrypt (pk^A, sk^A).

(d) calculates $r^c = \mathsf{e}_{pk^A}(r) = \mathsf{e}_{pk^A}(\mathsf{e}_{pk_l}(\mathsf{f}(M)))$

(e) sends (r^c, l) to the data owner.

3. **DataOwner**

(a) calculates $\mathsf{e}_{pk^A}(sk_l)$ and using it decrypts it's encryption part of r^c:

$$\begin{aligned}\mathsf{d}(r^c, \mathsf{e}_{pk^A}(sk_l)) &= \mathsf{d}(\mathsf{e}_{pk^A}(r), \mathsf{e}_{pk^A}(sk_l)) \equiv \mathsf{e}_{pk^A}(\mathsf{d}(r, sk_l)) \\ &\equiv \mathsf{e}_{pk^A}(\mathsf{d}(\mathsf{e}_{pk_l}(\mathsf{f}(M)), sk_l)) = \mathsf{e}_{pk^A}(\mathsf{f}(M)) = \tilde{r}\end{aligned}$$

(b) makes $\tilde{r}$ public.

4. **AlgorithmOwner**

(a) decrypts $\tilde{r}$ using its sk^A key.

$$\mathsf{d}(\tilde{r}, sk^A) = \mathsf{d}(\mathsf{e}_{pk^A}(\mathsf{f}(M)), sk^A) = \mathsf{f}(M)$$

In this protocol we assume that the auxiliary keys provided by the data owner are sufficient to perform all operations, $l < t$, but what happens if there are not, i.e., if $l > t$?. In this case the owner of the algorithm needs more auxiliary keys provided by the owner of the data. As the owner of the algorithm is not able to produce them, since this requires the private key of the last pair of auxiliary keys, the algorithm owner will have to ask the data owner to provide more auxiliary keys. This can be solved by generating a huge number of auxiliary keys by the data owner.

4.2 Second problem: how to verify that the decryption has been performed correctly?

The other main goal we wanted to achieve is to provide a secure service that does not depend on trust, i.e., that no one has to trust anyone for the process to be performed correctly while maintaining confidentiality and privacy at all times. Then, it would be desirable that without revealing the private inputs (the data M for the data owner and the algorithm f for the algorithm owner) to be able to verify somehow that the result obtained by the algorithm owner is precisely $\mathsf{f}(M)$.

Focusing only on this second problem, a first idea was to encrypt the final result with the public key with which the data owner encrypted the input data and check if it matches with what the data owner had passed. But since FHE cryptosystems, in order to be secure, have a part of randomness in the encryption function, this idea does not work. We proposed to share the randomness of the encryption, but this seriously damages the security of the system[II].

[II] By sharing the randomness part, anyone who sees it could encrypt all possible input data and do an intersection with the encrypted data provided by the data owner. Thus, anyone would get all the data in the clear.

4.2.1 Solution to the second problem: Homomorphic hashing

A hash function is a mathematical process that plays a fundamental role in public key cryptography. Among other things, it provides a way to guarantee data integrity and security against unauthorised modification. The problem we want to solve is exactly about this, so the idea we propose to solve this second problem is to use a special kind of hash function. Without going into formalities and details, a hash function is a versatile one-way cryptographic algorithm that maps an input of any size to a unique output of a fixed bit length. The resulting output, which is known as a hash or hash digest, is the resulting unique identifier. It means that, theoretically, no two pieces of different content have the same hash digest, or in other words, if the content changes, the hash digest changes as well. Hash functions serve as a checksum, or a way for someone to identify whether data has been tampered with. By using these functions, the message can be altered, but it gives us a way of knowing whether or not the content has been modified. This is because even the smallest of changes to a message results in the creation of a completely new hash value.

Let f be any circuit. In the problem we pose in this section, we need to check that by performing the double encryption, neither of the two parties can modify any data. One idea could be to hash the owner of the data to hash all the input data and the owner of the algorithm to hash the last one. With a normal hash function we would not be able to compare them, but what if the hash function had the homomorphic property? Imagine there is a hash function h that allows to calculate hashes homomorphically, that is,

$$\mathsf{h}(\mathsf{f}(m_1, \cdots, m_n)) \equiv \mathsf{f}((\mathsf{h}(m_1), \cdots, \mathsf{h}(m_n)).$$

We call h homomorphic hash function. This cryptographic primitive known as *homomorphic hashing*, was introduced by Bellare, Goldreich, and Goldwasser under the name incremental [49].

Homomorphic Hashing provides a neat solution to the problem of verifying data from untrusted peers, but it is not without its own drawbacks. It is complicated to implement and computationally expensive to compute, and to this date there are no well-stablished implementations, to the best of our knowledge. However we can theorically detail this idea a little bit more in depth so it can be used when homomorphic hashing is more advanced. We can then propose the following protocol, which is to be applied as a verification step after Protocol 1 has finished.

Protocol 2. Homomorphic Hash Scheme

Private inputs.
DataOwner. Sensitive data $M = \{m_1, ..., m_n\}$.
AlgorithmOwner. Algorithm f.

Public input. Homomorphic hashing h.

Goal. To verify the correctness of the encryption/decryption process after Protocol 1 has been applied.

The protocol:

1. **DataOwner**
   (a) encrypts each value of the data set M with the pk key: $\mathsf{e}_{pk}(m_i)$ for $\forall i \in \{1, \cdots, n\}$.
   (b) calculates each hash of the data set M with the homomorphic hash function h: $\mathsf{h}(m_i)$ for $\forall i \in \{1, \cdots, n\}$.
   (c) makes $h(m_i)$ public to the algorithm owner, for all i.
2. **Protocol 1 is performed**.
   The AlgorithmOwner obtains the result $d = \mathsf{f}(M)$.
3. **AlgorithmOwner**
   Since the algorithm owner knows f, h, d and $\mathsf{h}(m_i)$, they can calculate $\mathsf{h}(d)$ and $\mathsf{f}(h_1, \cdots, h_n)$.
   (a) If they are equivalent, then the algorithm owner knows that everything has been done correctly since

$$\mathsf{h}(d) = \mathsf{h}(\mathsf{d}(\mathsf{f}(e_1, \cdots, e_j))) \equiv \mathsf{f}(\mathsf{d}(\mathsf{h}(e_1), \cdots, \mathsf{h}(e_j)))) =$$
$$\mathsf{f}(\mathsf{h}(m_1), \cdots, \mathsf{h}(m_j))) = \mathsf{f}(h_1, \cdots, h_j).$$

   (b) If not, the algorithm owner concludes that something went wrong.

One of the disadvantages of this protocol is that the only part that is able to verify whether the process has been performed correctly or not is the owner of the algorithm themself, and therefore the trust must be placed on them. Therefore, we would say that it is a half solution to our problem, since we propose a way to verify if the process has been done well or not, but only one of the parties can make such verification.

Chapter 5

Solidity implementation

Ethereum is an open-source network and one of the largest public blockchain networks with an active community and a large public descentralized applications repository, that has been used for the implementation of smart contracts. The main elements of the smart contracts are functions, events, state variables, and modifiers and have been written in the high-level programming language known as Solidity [50]. We have used the test network Remix [51] to deploy smart contracts on the testnet. One of the drawbacks of Ethereum is that large data cannot be stored. The size of data that can be stored in a Ethereum's block is approximately 350 GB [52]. Therefore, it is difficult to store a large amount of data on the blockchain. For this project, we propose to use a peer-to-peer distributed file system that connects all computing devices to the same file system called Inter Planetary File System, IPFS [53]. It allows to data owners to upload their data and only the hash value returned from the IPFS has to be stored in a smart contract.

To define our protocol, in this chapter we will bring together the ideas we have explained so far. In the first section, we are going to take up what we have explained in the introduction and we are going to complete it with the solutions we have explained in the previous section. We are going to do this theoretically so that the protocol is better understood and in the following two sections the smart contract that we propose can be understood. We are going to make use of Ethereum testnet, although we will not create all the decentralised applications that we mentioned theoretically, we propose an example of a smart contract that would be valid in our protocol.

5.1 General idea

In our protocol we have two parties:

- Algorithm owners. They are those who have an algorithm that they do not want to be known by anyone. The results obtained after operating

with them are also known only by them. They need data from third parties.

- Data owners. They are those who are willing to share their data confidentially, that is, they leave their data encrypted, never in the clear.

Recall that the protocol allows for direct iteration of these two parties, whereby algorithm owners may use the encrypted data of others to operate on their algorithm and only the result of those operations is known to the owner of the algorithm themselves. It should be noted that neither the clear data neither the algorithms are made public, each party keeps its own information secret. Although there will be many users on each side, to simplify and thus better explain the protocol, we will focus on the case of a single algorithm owner. Taking this into account, we will explain the steps to follow in the protocol. For a better understanding follow the Figure 5.1.

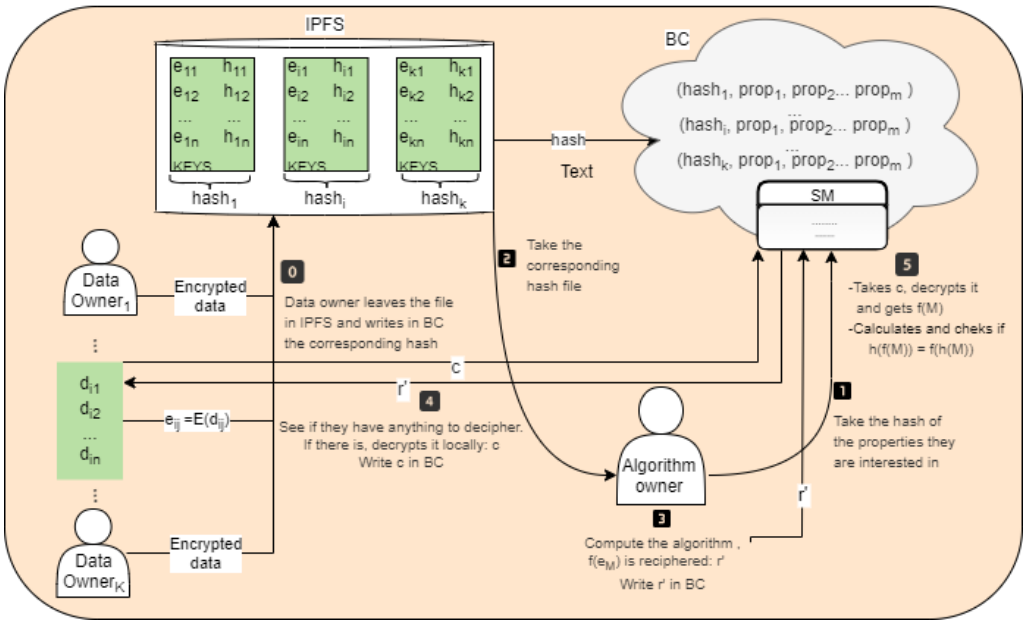

Figure 5.1: General idea: the steps of the protocol.

First of all, each data owner creates a file with their encrypted and hashed data, along with the used hash function and bootstrapping keys.

They drop it in the IPFS repository and get the hash of the file. That hash is the way we have to get to the file, so each owner must register that hash in the blockchain. In the record each one will indicate the properties of their data (for example, the age of the person to whom it belongs) (0). At any time the owner of the algorithm can query the hashes that are available in IPFS. Until they find a record with the properties they need, they must periodically query the blockchain. When they find what they are looking for, they take the corresponding hash of those properties (1). The iteration between the two parties starts in this step. Then the owner of the algorithm takes from IPFS the corresponding file (2). They operate their algorithm on the encrypted data. They encrypt the result obtained with their key and write the result (doubly encrypted) on the blockchain. In the register, they must indicate parameters such as which hash they are linked to, the public key they have used to encrypt and how many bootrsapping keys they have used, among other parameters (3). Each data owner can consult the ciphers that correspond to them for decryption. When a data owner has a cipher pending, using the information stored with the cipher of their choice, the double cipher is decrypted locally, remaining encrypted only with the key of the owner of the algorithm. Using the smart contract, they write the result obtained along with other identification parameters on the blockchain (4). The owner of the algorithm can consult the results registered on the blockchain. They choose one of them and, using the information provided by the corresponding data owner, they decrypt it locally. Finally, to verify if the result obtained is correct, they can calculate the hash of the result, using the hash of the IPFS file, and compute their algorithm on the hashed data of the file; and check if they match. If they match, the protocol has been performed correctly, otherwise something has gone wrong(5).

5.1.1 The protocol step by step

In the previous section we have explained the detailed general idea of our protocol. Building on what we have said in the previous section, in this section we show the flow of a possible smart contract that we have implemented.

In the following diagram (Figure 5.2) we have drawn the flow of the contract. That is to say, which steps each participant has to follow, which functions they have to execute at each moment, how they have to execute them... We have left the incentives system out, to make the process more visible but we will introduce it in the next section.

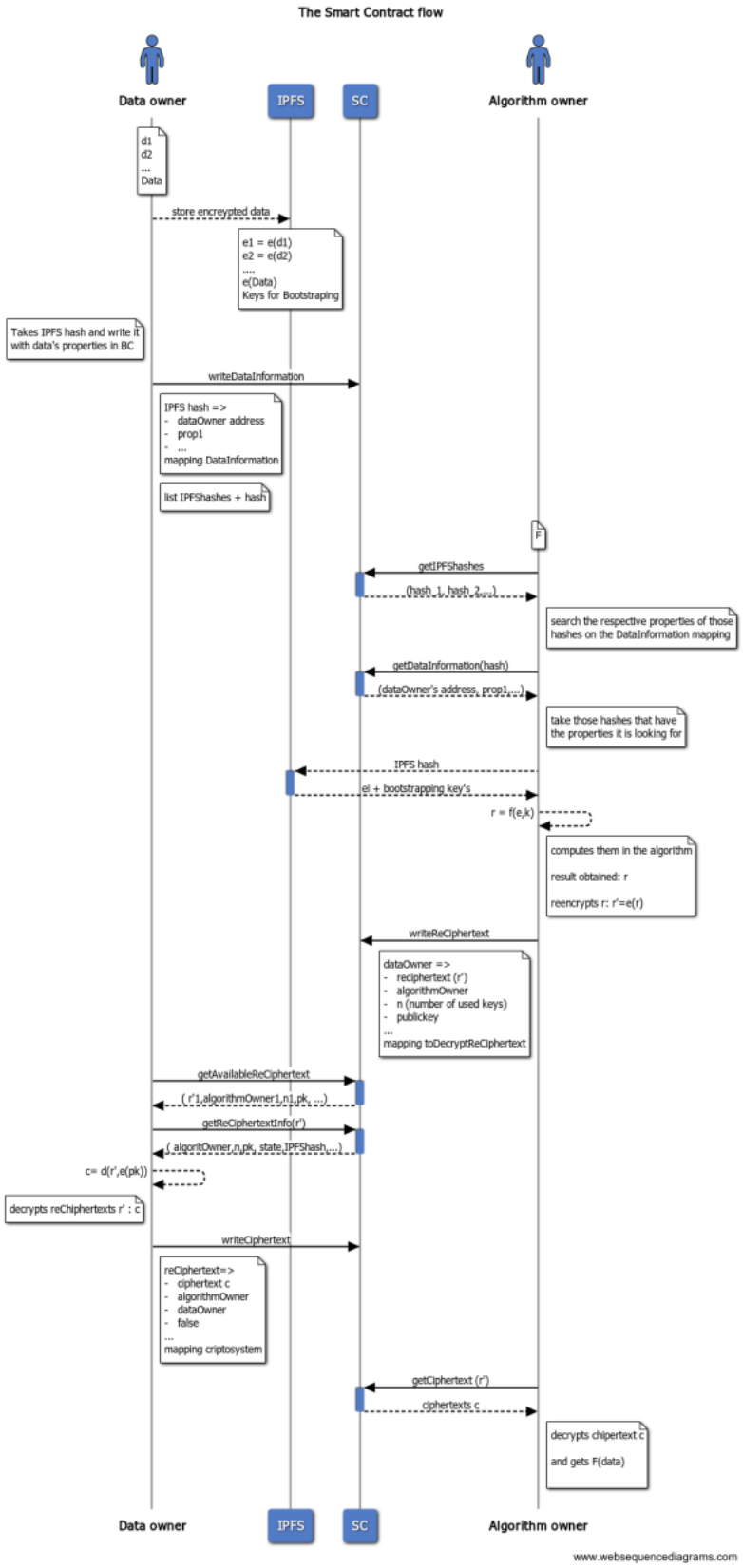

Figure 5.2: The main idea of the service

5.1.2 Protocol step by step with incentives

As we have seen in Chapter 4, the previous protocol has some drawbacks. On the one hand, we have no guarantee that both parties will complete the all the steps of the protocol. On the other hand, we do not have a method that tells us whether the protocol has been performed correctly or not, only the owner of the algorithm can verify it. To ensure as much as possible that both parties do their job properly, we have designed a system of incentives in which if the process is not completed, both parties lose. For this we have created two tokens, one with which the owners of the algorithms will pay their clients (owners of the data) for their services and the other to make a prestige ranking. This does not mitigate the problem, but it can reduce it.

(i) The data owner does the following:

   (a) Encrypts their data using a Fully Homomorphic Encryption system.

   (b) Stores their encrypted data in IPFS with several key pairs for bootstrapping. They take the hash of the document from IPFS. The hash can be seen as the key to the location of the encrypted data.

   (c) The hash together with the properties of the data is stored in the blockchain (writeDataInformation) so that any algorithm owner can access it.

(ii) Then the algorithm owner does the following:

   (a) Accesses the blockchain to query (getAvailableHash) for encrypted data and takes the hash of those that have the properties they are looking for the data (getDataInformation). Using this hash, they access the IPFS file and take the encrypted values and the keys for bootstrapping.

   (b) Compute the encrypted data in theis algorithm, by using the bootstapping method. We let l be the number of keys used. They store the number of keys used l.

   (c) Encrypt the result obtained with their public key and write the double cipher, the number of keys l, and its public public key to the blockchain. At the same time, they pay 12 data tokens (DT) to reward the data owner for facilitating the training of the algorithm (writeReCipher).

   (d) Before the owner of the data sees the reciphertetx (3.1), the algortithm owner can withdraw their 12 tokens and the reciphertext becomes invalid (AbortReCiphertext). This can be interesting if the algorithm owner thinks that too much time has passed and it is no longer useful for them to be deciphered.

(iii) Then the data owner does the following:

(a) Takes the reciphertext and pays 5 data token (in this way we push the owner of the data to follow the protocol). The time of this request is stored (getReCiphertextInfo). In total 17 tokens are stored.

(b) Decrypts its part locally, getting the encrypted value of applying the function to the data with the key of the algorithm owner. Writes the result on the blockchain. At this point the tokens are locked (17 tokens) and no one can abort the protocol(writeCiphertext).

(c) Does not write ciphertext.

(c.1) If one week has passed, automatically 12 data tokens are unlocked for the algorithm owner and 0 for data owner. The data owner is punished for offering a fake service (AbortTimePassed).

(c.2) If one week has not yet passed, the data owner can abort by unlocking 12 data tokens for the algorithm owner and 3 for the data owner (AbortReCiphertextByDataOwner) (this incentivates the data owner to decrypt it).

(iv) Then the algorithm owner does the following:

(a) is able to access the encryption (getCiphertext).

(a.1) If the owner of the algorithm confirms that it has obtained a valid result (payment(true)), it is understood that both participants has correctly completed the steps of the protocol, the locked tokens will be unlocked and 15 tokens will be sent to the data owner and 2 to the algorithm owner. In addition, the owner of the data will be rewarded with 3 reputation tokens (RT).

(a.2) If not (payment(false)), both participants will lose part of the tokens. 3 tokens will be sent to the data owner and 1 to the algorithm owner.

Note that in both cases we have given something back to the owner of the algorithm. This is because they are the only one who can confirm whether the protocol has been performed correctly or not and so we have designed a system in which it is more beneficial for them to indicate a response than to do nothing.

We have also introduced a new token called reputation token. Each data owner has a balance of how many processes they have started and how many reputation tokens they have. If the process has been done correctly, then it will get three tokens, but for whatever reason, if the process is not finished or is not processed correctly, it will not get any reputation tokens. The balance of each data owner can be queried using the function (getRTokenBalance).

It should be noted that an algorithm owner may interact with more than one data owner, and indeed an algorithm owner and a data owner may interact more than once. It is therefore not sufficient to relate algorithm owners to data owners. They must be related by reciphertext and IPFS hash too.

Payment functions

Finally, we have collected the functions involved in the incentive system. In the following figure we have indicated for each function the required input values and the value of the outputs.

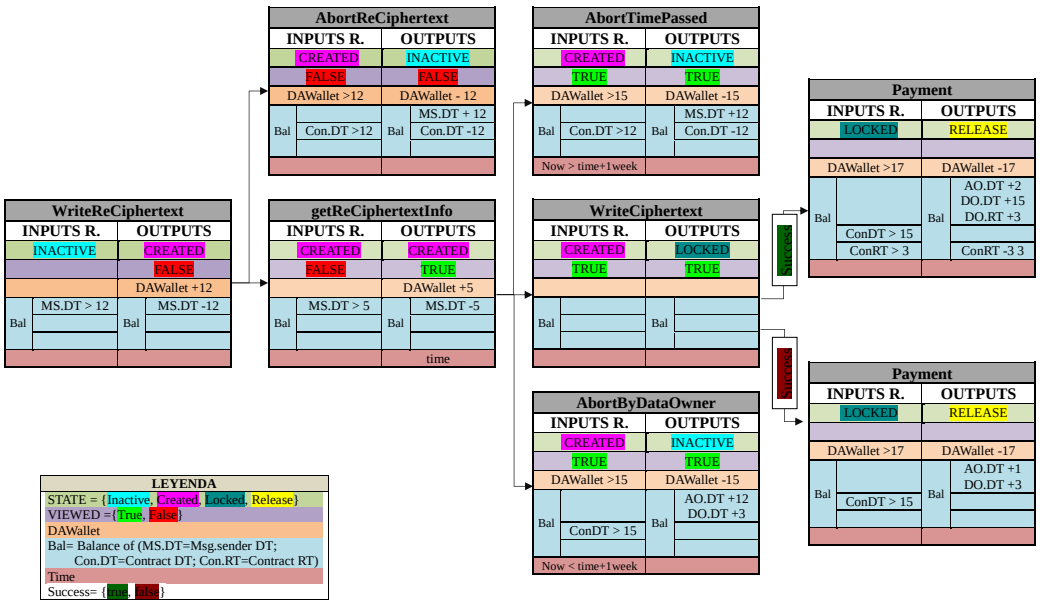

Figure 5.3: The main functions involved in the contract payment system

Chapter 6

Use cases

To conclude the work, in this chapter we will see two possible use cases provided by our protocol with which decentralised applications could be created.

The first use case was mentioned when introducing the protocol. The fact is that the protocol we propose fits very well in the world of research and in any field, be it finance, health, private companies... It is enough that there is a party that needs the results obtained from carrying out operations on certain data and another party willing to provide these data. For example, in the health field, researchers might want to analyse blood pressure, heart rate or glucose level to predict the likelihood of certain diseases. The more data they have, the better. The main advantage of this protocol is that it allows researchers to conduct more in-depth analyses as they can draw on data from people all over the world. In addition, researchers do not have to make public the algorithm they are using or the results obtained, which could be a problem if competitors knew about it. Furthermore, the data used in medicine is usually of a sensitive nature and therefore needs to be kept private. With the protocol we propose, the data is encrypted while the researchers can carry out their research. Thanks to the anonymity provided by the blockchain, users do not have to disclose their identities, nor the data itself at any time.

The other use case we propose is about using Google's infinite amount of data to create, sell or improve products. In the age of the digital economy, both the methods for analysing and processing 'big data' and the knowledge gained from it have become real assets for companies. The analysis of the information obtained from the network allows to have a better understanding of the essential issues of a business (consumption patterns in its relevant market, customer needs, etc.) and to direct resources in a very efficient and profitable way. In fact, the knowledge gained from data analysis has become decisive in the decision making and strategy design of many of the most sophisticated companies. Precisely for this reason, the tools to analyse

and process large amounts of data and the ability to interpret them form the basis of the business and the competitive advantage of many companies. The very importance of these intangible assets makes their protection a crucial element for the success and consolidation of these businesses. We cannot lose sight of the fact that the disclosure of an algorithm or free access to the knowledge stored in the databases generated may jeopardise a business based on the exploitation of these intangibles. Therefore, if Google were to store the (encrypted) data it has in files, companies could use its algorithms to see what people need in order to create, modify or sell products by area. Having Blockchain technology as a base makes it easier, thanks to its anonymity and borderless properties, to make it accessible to any company that is interested, without discrimination, as any company in the world could participate. In this case, the decentralisation property offered by the blockchain would not be fully exploited, as the data depends on Google, but it could be considered that the users themselves could provide the same information or make them fill in a questionnaire.

Chapter 7

Conclusion

In this chapter we summarise the contributions presented in this work, draw some conclusions and outline some open questions.

In this dissertation, we

- review the state of the art of homomorphic cryptosystems and we give the basic notions to understand these encryption schemes;
- look at the history of Blockchain technology and its most relevant features. We focus on Ethereum and we develop a smart contract proposal for our protocol; and
- making use of these two great tools we present a decentralised, borderless and scalable system for data processing while preserving the privacy of both the data and the algorithms and knowledge acquired through them

Personal data, and sensitive data in general, should not be trusted to third parties, which are susceptible to attacks and unauthorised use. Instead, users must own and control their data without compromising security or limiting the ability of companies and authorities to offer personalised services. In addition, companies, public or private, whose business relies on methods to analyse and process data as well as the knowledge gained from it, need protection for these assets. Our protocol covers these requirements by combining a blockchain, used as a platform for interaction between participants, with a homomorphic encryption scheme that allows to operate with the encrypted data.

Finally, we raise a couple of questions that we leave open and that can serve as a guide for future research.

(i) We have tried to find a complete solution to the Problem 2 but we have not achieved this goal. The solution we propose covers only half of the problem and is also far from being practically realistic since no implementation is known that meets the necessary requirements. More

seriously, the solution we propose may leave a security gap, since with the hashed data, and the hash function used, a "dictionary attack" could be performed. Therefore, we leave open the problem of finding a practical complete solution.

(ii) As in many cases, in our protocol we assume the integrity of the data. As the data is anonymised and encrypted, this could be a problem as it could be forged by anyone. Therefore, the other door we leave open is to find a way for the reliability of the input data.

Appendix A

Smart Contract

```
//https://docs.openzeppelin.com/contracts/4.x/access-control
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.0/contracts/token/ERC20/ERC20.sol";

contract dataToken is ERC20 {

    constructor() ERC20("dataToken", "DT") {
        _mint(_msgSender(), 100000* (10** uint256(decimals())));
    }
}
```

```
//https://docs.openzeppelin.com/contracts/4.x/access-control
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.0/contracts/token/ERC20/ERC20.sol";

contract reputationToken is ERC20 {
       constructor() ERC20("reputationToken", "RT") {
        _mint(_msgSender(), 1000);
    }
}
```

Figure A.1: dataToken and reputationToken smart contract

```
//SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

library removeCiphertext {

    function luzera(bytes32[] memory array) external pure returns(uint){
        return(array.length);
    }

    function remove(bytes32[] storage array, uint index) public {
        require(index <= array.length, "mal indice");

        for (uint i = index; i<array.length-1; i++){
            array[i] = array[i+1];
        }
        delete array[array.length-1];

        array.pop();
    }
}
```

Figure A.2: Library that deletes an element given a list and an index

```
/*
     Write and read information from this SC
     Automatic and completely transparent payment

example of bytes32:   0x1465737400000000000000000000000000000000000000000000000000000000
*/

//SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

import "./dataToken.sol";
import "./reputationToken.sol";
import "./removeCiphertext.sol";
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.0/contracts/access/AccessControl.sol";

contract writing4 is AccessControl {

    //These addresses have to be changed if any of these contracts are modified.
    address private DTokenAddress = 0xed27012c24FDa47A661De241c4030ecB9D18a76d;
    address private RTokenAddress = 0x16500370A61d015f025e4C74dAdb972042567d9a;

    dataToken private DToken;
    reputationToken private RToken;

    constructor(){
        DToken = dataToken(DTokenAddress);
        RToken = reputationToken(RTokenAddress);
        _setupRole(OWNER, _msgSender());
    }

    using removeCiphertext for bytes32[];

    bytes32 internal constant OWNER = keccak256("OWNER"); // address who created this contract
    bytes32 internal constant ALGORITHMOWNER= keccak256("ALGORITHMOWNER");      //address type ALGORITHMOWNER
    bytes32 internal constant DATAOWNER = keccak256("DATAOWNER");               //address type DATAOWNER

struct Properties {                                         //Struct to store the properties of the data stored in IPF
    address dataOwner;                                             //address direction who writes the properties
    uint age;                                                      //age of data owner
    uint weight;                                                   //weight of data owner
    uint height;                                                   //height of data owner
    string nationality;                                            //nationality of data owner
}

mapping(bytes32 => Properties) dataInformation;        //mapping that relates the IPFS hash to the respective properti

bytes32[] IPFShashes;                                 //list of hashes corresponding to the data available in IPFS



struct reCiphertext {                                        // Structure to store
        address algorithmOwner;                                     //the AlgorithmOwner that calculated the reCipher
        address dataOwner;                                          //address direction who belongs the data
        uint n;                                                     //indicates the number of bootstrapping keys used
        bytes publicKey;                                            //Public key used by the algorithm owner to reenc
        bool viewed;                                                //boolean variable indicating whether the reciphe
        State state;                                                //state that indicate the status of the lockout.
        bytes32 IPFShash;                                           //to indicate which json file it belongs to.
        uint time;                                                  //variable to indicate the time of reciphertext w
    }

    enum State {Inactive, Created, Locked, Release}     /* Enumerator indicates the status of the lockout
                                                                Inactive : the process has not started or has been
                                                                Created  : The client has blocked the tokens but c
                                                                Locked   : the provider has written the decryption
                                                                           until it is verified that the decrypti
                                                                Release  : tokens have been released. */


mapping(bytes => reCiphertext) RecipherInfo;            //mapping that relates the reciphertext with its information.
mapping(address => bytes[]) toDecryptReCiphertext;      //mapping that indicates to each dataowner which reciphertexts
struct criptoSystem {                                   //Structure for storing
    bytes ciphertext;                                        //the ciphertext,
    address algorithmOwner;                                  //the Algorithm owner who requested it
    address dataOwner;                                       //and the data owner who calculated it
    //bool verified;                                         //The strucure also has a boolean that indicates if the c
}

mapping(bytes => criptoSystem) criptosystem;            //mapping that relates the reciphertext with its criptosystem
mapping(address => mapping( address => uint)) DAwallet; //mapping to store de amount of tokens are in the data-algorit

mapping(address => uint) ratio;                        //mapping to calculate token/(process started) ratio
```

Figure A.3: First part

```
    modifier OnlyAlgorithmOwner(address _sender) {
        require( hasRole(ALGORITHMOWNER, _sender), "Only algorithm owners can call this function");
        _;
    }

    modifier OnlyThisAlgorithmO(bytes memory _reciphertext){
        require(RecipherInfo[_reciphertext].algorithmOwner == msg.sender, "You are not the owner of this cipher
        _;
    }

    modifier OnlyDataOwner(address _sender) {
        require( hasRole(DATAOWNER, _sender), "Only data owners can call this function");
        _;
    }

    modifier OnlyThisDataO(bytes memory _reciphertext){
        require(RecipherInfo[_reciphertext].dataOwner == msg.sender, "You dont have the access to this cipherte
        _;
    }

    modifier inState(State _state, bytes memory _reciphertext) {
        require(RecipherInfo[_reciphertext].state == _state, "The state is invalid.");
        _;
    }

    modifier condition(bool _condition){
        require (_condition, "does not satisfy the condition");
        _;
    }

    modifier ContractEnoughDT (uint amount){
        require(DToken.balanceOf(address(this)) >= amount, "there is no enough tokens in this contract");
        _;
    }

    modifier ContractEnoughRT (uint amount){
        require(RToken.balanceOf(address(this)) >= amount, "there is no enough tokens in this contract");
        _;
    }

    modifier enoughDT (uint amount) {
        require(DToken.balanceOf(msg.sender) >= amount, "You do not have enough tokens to continue");
        _;
    }


    event NewData(bytes32);
    event NewReCiphertext();
    event Decrypted(address);
    event AbortedReCiphertext(bytes);


/*********************************functions*************************************/

//function to write the corresponding properties of a hash
    function writeDataInformation(bytes32 _hash, uint _age, uint _weight, uint _height, string calldata _nation
        public {
        dataInformation[_hash] = Properties(msg.sender, _age, _weight, _height, _nation);
        IPFShashes.push(_hash);
        emit NewData(_hash);
        _setupRole(DATAOWNER, msg.sender);
        }

//function to view all available hashes. Anyone can access to it.
    function getAvailableHash() public view returns(bytes32[] memory) {
        return IPFShashes;
    }

//function to remove available IPFSs data
//This function could be interesting for the dataOwner if they do not want their data to be operated anymore.
    function removeIPFSHash(uint _index) public{
        require(dataInformation[IPFShashes[_index]].dataOwner==msg.sender, "you do not have permission to delet
        require(IPFShashes.length >= _index, "Invalid index, check IPFSHash list with getAvailableHash function
        dataInformation[IPFShashes[_index]]=Properties(0x0000000000000000000000000000000000000000,0,0,0, "");
        removeCiphertext.remove(IPFShashes, _index);
    }
```

Figure A.4: Second part

```
//function to view the corresponding properties of a concrete hash. Anyone can access to it.
    function getDataInformation(bytes32 _hash) public view
    returns(address _dataOwner, uint _age, uint _weight, uint _height, string memory _nation) {
        return (dataInformation[_hash].dataOwner, dataInformation[_hash].age, dataInformation[_hash].weight,
        dataInformation[_hash].height, dataInformation[_hash].nationality);
    }

//function to indicate the ratio of a dataOwner.
//So that the AlgorithmOwner calls this function evaluates whether to continue with the process or not.
//OnlyAlgorithmOwner(msg.sender)
    function getRTokenBalance(address _dataOwner) public view returns(uint, uint){
        require(hasRole(DATAOWNER, _dataOwner), "this address is not allowed to hold RT tokens");
        return(RToken.balanceOf(_dataOwner), ratio[_dataOwner]);
    }

//function to write a reciphertext along with its properties using mapping. Locks 12tokens from the caller(algorithOwner)
//Set up to the message sender an ALGORITHMOWNER role.
    function writeReCipher(bytes memory _reciphertext, address _dataOwner, uint _n, bytes calldata _pk, bytes32 _IPFShash) public
    enoughDT(12)
    {
        DToken.transferFrom(msg.sender, address(this), 12);
        DAwallet[_dataOwner][msg.sender] += 12;
        RecipherInfo[_reciphertext]=reCiphertext(msg.sender,_dataOwner, _n, _pk, false, State.Created,  _IPFShash, 0);
        toDecryptReCiphertext[_dataOwner].push(_reciphertext);
        _setupRole(ALGORITHMOWNER, msg.sender);
        emit NewReCiphertext();
    }

//algortithmOwner can withdraw its 12 tokens and the reciphertext becomes invalid.
//It can be interesting if the AlgorithmOwner thinks that too much time has passed and it is no longer useful for him to be deci
    function AbortReCiphertext(bytes calldata _reciphertext) public
    OnlyThisAlgorithmO(_reciphertext) inState(State.Created, _reciphertext) ContractEnoughDT(12)
    {
        require(!RecipherInfo[_reciphertext].viewed, "dataOwner could be doing operations");
        require(DAwallet[RecipherInfo[_reciphertext].dataOwner][msg.sender] >=12, "not enough tokens are lockout");
        RecipherInfo[_reciphertext].state = State.Inactive;
        emit AbortedReCiphertext(_reciphertext);
        DAwallet[RecipherInfo[_reciphertext].dataOwner][msg.sender] -= 12;
        DToken.transfer(msg.sender, 12);
    }

//function to view all available reCiphers to decrypt for a particular data owner. Only dataOwner's can access
    function getAvailableReCipher() public view OnlyDataOwner(msg.sender) returns(bytes[] memory) {
        return toDecryptReCiphertext[msg.sender];
    }


//function to obtain additional information needed for decryption. Only the corresponding data owner can call this function.
//DataOwner locks 5 tokens. Time starts countig.
    function getReCiphertextInfo(bytes memory _reciphertext) external
    OnlyThisDataO(_reciphertext) enoughDT(5) inState(State.Created,_reciphertext)
    returns(address _algorithmOwner, uint _n, bytes memory _publicKey, bool _viewed,bytes32 _IPFShash) {
        require(!RecipherInfo[_reciphertext].viewed, "the ciphertext has already been consulted");
        DToken.transferFrom(msg.sender, address(this), 5);
        DAwallet[msg.sender][RecipherInfo[_reciphertext].algorithmOwner] += 5;
        RecipherInfo[_reciphertext].viewed = true;
        RecipherInfo[_reciphertext].time =block.timestamp * 1 weeks;
        ratio[msg.sender] += 1;
        return (RecipherInfo[_reciphertext].algorithmOwner, RecipherInfo[_reciphertext].n, RecipherInfo[_reciphertext].publicKey,
        RecipherInfo[_reciphertext].viewed, RecipherInfo[_reciphertext].IPFShash);
    }

//Data owners can, after seeing the reciphertext they have to decrypt, cancel the process during that week.
    function AbortByDataOwner(bytes memory _reciphertext) external
    OnlyThisDataO(_reciphertext) ContractEnoughDT(15) inState(State.Created,_reciphertext)
    {
        require(RecipherInfo[_reciphertext].time + 1 weeks > block.timestamp, "too much time has passed to use this function");
        require(DAwallet[msg.sender][RecipherInfo[_reciphertext].algorithmOwner] >=15);
        require(RecipherInfo[_reciphertext].viewed, "Previously you have not locked the tokens." );
        DAwallet[msg.sender][RecipherInfo[_reciphertext].algorithmOwner] -= 15;
        RecipherInfo[_reciphertext].state= State.Inactive;
        //RecipherInfo[_reciphertext].viewed == false;
        DToken.transferFrom( address(this),RecipherInfo[_reciphertext].algorithmOwner, 12);
        DToken.transferFrom( address(this),msg.sender, 3);
    }
//Data owners have one week to decrypt the reencryption.
//Otherwise the owner of the algorithm can cancel the order and withdraw the money.
    function AbortTimePassed(bytes memory _reciphertext) external
    OnlyThisAlgorithmO(_reciphertext) ContractEnoughDT(12) inState(State.Created,_reciphertext)
    {
        require(RecipherInfo[_reciphertext].time + 1 weeks < block.timestamp, "not enough time has passed.");
        require(DAwallet[msg.sender][RecipherInfo[_reciphertext].algorithmOwner] >=15);
        require(RecipherInfo[_reciphertext].viewed, "Previously you have not locked the tokens." );
        DAwallet[msg.sender][RecipherInfo[_reciphertext].algorithmOwner] -= 15;
        RecipherInfo[_reciphertext].state= State.Inactive;
        DToken.transferFrom( address(this),RecipherInfo[_reciphertext].algorithmOwner, 12);
    }
```

Figure A.5: Third part

```
//function to write ciphertext with reciphertext using mapping.
    function writeCiphertext(bytes memory _ciphertext, bytes memory _reciphertext) public
    OnlyThisData0(_reciphertext)
    condition(RecipherInfo[_reciphertext].viewed)
    inState(State.Created, _reciphertext)
    {
        criptosystem[_reciphertext]= criptoSystem(_ciphertext, RecipherInfo[_reciphertext].algorithmOwner, msg.sender);
        emit Decrypted(RecipherInfo[_reciphertext].algorithmOwner);
        RecipherInfo[_reciphertext].state=State.Locked;
        }


//function in which the owner of the algorithm gets the ciphertext.
    function getCiphertext(bytes memory _reciphertext) public view
    OnlyThisAlgorithm0(_reciphertext) returns(criptoSystem memory)
    {
        return criptosystem[_reciphertext];
    }

//function in which the owner of the algorithm indicates whether the process was successful or not.
//Depending on that, the locked tokens will be unlocked in one way or another.
    function payment(bool _success, bytes memory _reciphertext) public
    OnlyThisAlgorithm0(_reciphertext) inState(State.Locked, _reciphertext) ContractEnoughDT(17) ContractEnoughRT(3)
    condition(DAwallet[RecipherInfo[_reciphertext].dataOwner][RecipherInfo[_reciphertext].algorithmOwner]>=17)
    {
        RecipherInfo[_reciphertext].state=State.Release;
        DAwallet[RecipherInfo[_reciphertext].dataOwner][RecipherInfo[_reciphertext].algorithmOwner] -= 17;
        if (_success){
            DToken.transfer(RecipherInfo[_reciphertext].algorithmOwner, 2);
            DToken.transfer(RecipherInfo[_reciphertext].dataOwner, 15 );
            RToken.transfer(RecipherInfo[_reciphertext].dataOwner, 3);
        } else {
            DToken.transfer(RecipherInfo[_reciphertext].algorithmOwner, 1);
            DToken.transfer(RecipherInfo[_reciphertext].dataOwner, 3 );
        }
    }
    function faucet(uint _amount) external {
        DToken.transfer(msg.sender, _amount);
    }

    function getDAwallet(address _data0, address _algorithm0) external view returns(uint){
        return(DAwallet[_data0][_algorithm0]);
    }

}
```

Figure A.6: Last part

Bibliography

[1] ScienceDaily. Big data, for better or worse: 90% of world's data generated over last two years. 2013. Available online: https://www.sciencedaily.com/releases/2013/05/130522085217.htm

[2] Bhageshpur, K. Council Post: Data Is The New Oil – And That's A Good Thing. Available online: https://www.forbes.com/sites/forbestechcouncil/2019/11/15/data-is-the-new-oil-and-thats-a-good-thing/

[3] Bentein, A. Data is the New Gold — QAD Blog. Available online: https://www.qad.com/blog/2021/04/data-is-the-new-gold

[4] Homomorphic encryption - Wikipedia. Available online: https://en.wikipedia.org/wiki/Homomorphic_encryption

[5] Zama. Available online: https://zama.ai/

[6] iDASH secure genome analysis competition 2018. Available online: https://bmcmedgenomics.biomedcentral.com/articles/10.1186/s12920-020-0715-0

[7] Hackeo masivo de Quora: Roban 100 millones de cuentas del Yahoo respuestas inglés. (2018). Available online: https://as.com/meristation/2018/12/04/betech/1543928574_676934.html

[8] Pastor, J. (2021). Robo masivo de datos en Facebook: los datos personales de 533 millones de usuarios se filtran online. Available online: https://www.xataka.com/seguridad/robo-masivo-datos-facebook-datos-personales-533-millones-usuarios-se-filtran-online

[9] Privacy and blockchain - Wikipedia. Available online: https://en.wikipedia.org/wiki/Privacy_and_blockchain

[10] Accounting - Wikipedia. Available online: https://en.wikipedia.org/wiki/Accounting

[11] The untold history of Bitcoin: Enter the Cypherpunks. (2021). Available online: https://medium.com/swlh/the-untold-history-of-bitcoin-enter-the-cypherpunks-f764dee962a1

[12] What is the Cypherpunks Mailing List?. Available online: https://cryptoanarchy.wiki/getting-started/what-is-the-cypherpunks-mailing-list

[13] Blockchain - Wikipedia. Available online: https://en.wikipedia.org/wiki/Blockchain

[14] Bayer, D., Haber, S., & Stornetta, W. S. (1993). Improving the Efficiency and Reliability of Digital Time-Stamping. Sequences II, 329-334. doi:10.1007/978-1-4613-9323-8_24. Available online: https://www.math.columbia.edu/~bayer/papers/Timestamp_BHS93.pdf

[15] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008). Available online: https://bitcoin.org/en/bitcoin-paper

[16] Understanding Blockchain Consensus Models. Persistant (2017). Available online: https://www.persistent.com/wp-content/uploads/2017/04/WP-Understanding-Blockchain-Consensus-Models.pdf

[17] Blockchain Explained. Investopedia. Available online: https://www.investopedia.com/terms/b/blockchain.asp

[18] Gupta, M. Blockchain for dummies (3rd ed.). Hoboken, New Jersey: John Wiley & Sons, Inc. Available online: https://www.ibm.com/downloads/cas/OK5M0E49

[19] Nofer, M., Gomber, P., Hinz, O., & Schiereck, D. (2017). Blockchain. Business & Information Systems Engineering, 59(3), 183-187. doi:10.1007/s12599-017-0467-3 Available online: https://link.springer.com/content/pdf/10.1007/s12599-017-0467-3.pdf

[20] Hash function - Wikipedia. Available online: https://en.wikipedia.org/wiki/Hash_function

[21] Chris, J. Blockchain: Background and Policy Issues; Congressional Research Service, 2018. Available online: https://www.hsdl.org/?abstract&did=808684

[22] What are the 4 different types of blockchain technology?. Available online: https://searchcio.techtarget.com/feature/What-are-the-4-different-types-of-blockchain-technology

[23] UTXO vs. Account Model, Horizen Academy. Available online: https://academy.horizen.io/technology/expert/utxo-vs-account-model/

[24] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project Yellow Paper, 2014. Available online: https://gavwood.com/paper.pdf

[25] Turing completeness - Wikipedia. Available online: https://en.wikipedia.org/wiki/Turing_completeness

[26] Szabo, N. (1997). Formalizing and Securing Relationships on Public Networks. First Monday, 2(9). https://doi.org/10.5210/fm.v2i9.548 Available online: https://firstmonday.org/ojs/index.php/fm/article/view/548/469

[27] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. white paper, 3(37), 2014. Available online: https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf

[28] Ethereum Virtual Machine (EVM), ethereum.org. Available online: https://ethereum.org/en/developers/docs/evm/

[29] "Ethereum Homestead Documentation — Ethereum Homestead 0.1 documentation". Available online: http://www.ethdocs.org/.

[30] Armknecht, F., Boyd, C., Carr, C., Gjøsteen, K., Jaschke, A., A. Reuter, C., & Strand, M. (2015). A Guide to Fully Homomorphic Encryption. Available online: https://eprint.iacr.org/2015/1192.pdf

[31] Acar, A., Aksu, H., Uluagac, S. A., & Conti, M. (2017). A Survey on Homomorphic Encryption Schemes: Theory and Implementation.arXiv:1704.03578v2. Available online: https://dl.acm.org/doi/pdf/10.1145/3214303

[32] Performance analysis of modified RSA and RSA homomorphic encryption scheme for cloud data security, Researchgate. Available online: https://www.researchgate.net/publication/314103835_Performance_analysis_of_modified_RSA

[33] Paillier cryptosystem - Wikipedia. Available online: https://en.wikipedia.org/wiki/Paillier_cryptosystem

[34] Rivest, Adleman, Dertouzos (n.d.). ON DATA BANKS AND PRIVACY HOMOMORPHISMS. Available online: http://people.csail.mit.edu/rivest/RivestAdlemanDertouzos-OnDataBanksAndPrivacyHomomorphisms.pdf

[35] M. V. Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan (n.d.). Fully Homomorphic Encryption over the Integers. Available online: https://eprint.iacr.org/2009/616.pdf

[36] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in IEEE Transactions on Information Theory, vol. 31, no. 4, pp. 469-472, July 1985, doi: 10.1109/TIT.1985.1057074. Available online: https://caislab.kaist.ac.kr/lecture/2010/spring/cs548/basic/B02.pdf

[37] Goldwasser–Micali cryptosystem. Shafi Goldwasser and Silvio Micali's. Available online: https://en.wikipedia.org/wiki/Goldwasser%E2%80%93Micali_cryptosystem

[38] Dan Boneh, Eu-Jin Goh and Kobbi Nissim, "Evaluating 2-DNF Formulas on Ciphertexts". Available online: https://crypto.stanford.edu/ dabo/papers/2dnf.pdf

[39] Gentry C (2009) A fully Homomorphic encryption scheme. In: Stanford University. Stanford, PhD Thesis. Available online: https://crypto.stanford.edu/craig/craig-thesis.pdf

[40] Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (n.d.). Fully Homomorphic Encryption without Bootstrapping. Available online: https://eprint.iacr.org/2011/277.pdf

[41] Gentry, C., Sahai, A., & Waters, B. (n.d.), "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based". Available online: https://eprint.iacr.org/2013/340.pdf

[42] Wade, J. A., "An Expos e on the Lopez-Alt-Tromer-Vaikuntanathan Fully Homomorphic Encryption Scheme". Available online: https://web.wpi.edu/Pubs/E-project/Available/E-project-042513-000009/unrestricted/An_Expose_on_the_LTV_FHE_Scheme.pdf

[43] Cheon, J. H., Kim, ,., Kim, M., Song, Y. (n.d.). Homomorphic Encryption for Arithmetic of Approximate Numbers. Available online: https://eprint.iacr.org/2016/421.pdf

[44] Chillotti, I., Gama, N., Georgieva, M., Izabachene, M. (n.d.). TFHE: Fast Fully Homomorphic Encryption over the Torus. Available online: https://eprint.iacr.org/2018/421.pdf

[45] Functional completeness. (2021, June 14). Available online: https://en.wikipedia.org/wiki/Functional_completeness

[46] Zvika Brakerski and Vinod Vaikuntanathan. "Efficient Fully Homomorphic Encryption from (Standard) LWE". Available online: https://eprint.iacr.org/2011/344.pdf

[47] Canetti, R., Richelson, S., Vaikuntanathan, V., Raghuraman, S. (n.d.). Chosen-Ciphertext Secure Fully Homomorphic Encryption. Available online: https://link.springer.com/chapter/10.1007/978-3-662-54388-7_8

[48] Fillinger, M. Master of Logic Project Report: Lattice Based Cryptography and Fully Homomorphic Encryption (2012, August 18) Available online: https://homepages.cwi.nl/ schaffne/courses/reports/-MaxFillinger_FHE_2012.pdf

[49] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. "Incremental cryptography: The case of hashing and signing". Available online: https://cseweb.ucsd.edu/ mihir/papers/inc1.pdf

[50] Solidity. Available online: https://docs.soliditylang.org/en/v0.8.7/

[51] Ethereum IDE community. Available online: https://remix-project.org/

[52] Decentralized Storage. Available online: https://ethereum.org/en/developers/docs/storage/

[53] IPFS Powers the Distributed Web. Available online: https://ipfs.io/