UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIA DA COMPUTAÇÃO

Anthony Bernardo Kamers

Homomorphic Encryption: Introduction and Applicabilities

Florianópolis
2023

Anthony Bernardo Kamers

Homomorphic Encryption: Introduction and Applicabilities

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de Graduação em Ciência da Computação.

Florianópolis, 12 de dezembro de 2023.

Prof. Lúcia Helena Martins Pacheco, Dr[a]
Coordenadora do Curso

Banca Examinadora:

Prof[a]. Thaís Bardini Idalino, Dr[a].
Orientadora
Universidade Federal de Santa Catarina

Prof. Ricardo Felipe Custódio, Dr.
Avaliador
Universidade Federal de Santa Catarina

Prof. Daniel Panario, Dr.
Avaliador
Carleton University, Canada

Anthony Bernardo Kamers

Homomorphic Encryption: Introduction and Applicabilities

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciência da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Ciência da Computação.
Orientadora: Prof[a]. Thaís Bardini Idalino, Dr[a].
Coorientador: Gustavo Zambonin

Florianópolis
2023

À minha família e à minha mulher perfeitas

ACKNOWLEDGEMENTS

Agradeço aos meus pais, Márcio e Márcia, por sempre acreditar em minhas capacidades, me ajudando com muito mais do que precisava, e fazer de mim a pessoa que sou hoje. Agradeço ao meu irmão Bruno, por sempre ser meu companheiro ao estudar, trabalhar, ir à academia, jogar, ou fazer qualquer coisa juntos. Agradeço à minha namorada, por estar comigo nesta jornada e me apoiar em todos os momentos, mesmo quando não encontrava forças dentro de mim para continuar. Vocês são a minha família e os amo para sempre!

Agradeço à minha orientadora Thaís, por sempre demonstrar sua paixão pelo seu trabalho e pesquisa, sempre me instigando a melhorar meu estudo e a buscar novos horizontes. Agradeço aos meus companheiros de trabalho e pesquisa no LabSEC, em especial ao meu coorientador, Zambonin, que não mede esforços por seus amigos, nem pela ciência; o agradeço também por sempre poder contar com seu apoio e amizade, sempre compartilhando os fatos mais inusitados, sobre as mais diversas áreas com seu jeito único e distinto. Agradeço também ao prof. Daniel Panário por poder compartilhar um pouco de sua vasta experiência em prol deste trabalho.

“Crypto will not be broken, it will be bypassed.”
Adi Shamir

RESUMO

A criptografia homomórfica é um mecanismo que permite fazer operações diretamente no texto cifrado, ou seja, sem a necessidade de descriptografá-lo previamente. A propriedade de homomorfismo em criptografia é de interesse na área desde que foi mencionada em 1978 (RIVEST; SHAMIR; ADLEMAN, 1978), onde foi proposto um possível esquema de encriptação que continha a propriedade mencionada. Com isso, várias seriam as aplicações, pois não seria mais necessário fazer um esquema de troca de chaves privadas, mascaramento de determinadas informações no fluxo de transferência de dados, não haveria mais necessidade de descriptografar o texto cifrado para fazer operações sobre ele, entre outras. Para se fazer isso, é necessário usar o conceito matemático de funções homomórficas, dando assim origem ao nome "criptografia homomórfica". Como exemplo claro de utilização desse tipo de encriptação, pode-se citar o processamento de informações por computação em nuvem, dado que, para fazer o cômputo dos dados é necessário descriptografar os mesmos, podendo expor elementos sigilosos a algum ataque. Esse problema seria sanado utilizando operações sobre o texto cifrado com criptografia homomórfica. Este trabalho tem como objetivo fazer um estudo introdutório à criptografia homomórfica, seus tipos mais básicos e suas aplicabilidades em diferentes áreas da computação. Para isso, será feito um estudo de materiais científicos acerca do estado da arte, para entendimento dos algoritmos de algumas variantes e suas principais aplicações na prática. Como resultado, terá um material que permite a compreensão das versões mais básicas de criptografia homomórfica, assim como suas aplicações, estimulando o uso em sistemas que manipulam dados pessoais.

Keywords: criptografia. criptografia homomórfica. homomorfismo. segurança da computação. anonimização de dados.

ABSTRACT

Homomorphic encryption is a mechanism that allows people to operate directly over the ciphered text, which means, without the need to decrypt it first. The property of homomorphic encryption is very interesting for cryptography since it was first mentioned in 1978 (RIVEST; SHAMIR; ADLEMAN, 1978), where was proposed a possible encryption scheme, containing the mentioned property. Several applications would be possible, because it would not be necessary to make a private key exchange scheme, masking some information on the flow of data transference, needing to decrypt the ciphered text to make operations onto it, among other applications. To make that, it is necessary to use the mathematical concept of homomorphic functions, giving its name "homomorphic encryption". As a clear example of using this encryption scheme, it is possible to cite the processing of information in cloud computing, given that, to make the data computation possible, it is necessary to decrypt it, which can expose confidential elements to an attack. This problem would be avoided by using operations over the ciphered text with homomorphic encryption. This work has as a goal to make an introductory study of homomorphic encryption, its basic variants, and applications in different fields of computation. In order to do that, we will perform a study on scientific published material on the state of the art for some variants, and their main applications in practice. As a result, we will have material that allows the comprehension of basic versions of homomorphic encryption schemes, as their applications, stimulating the usage of systems that manipulate personal data.

Keywords: cryptography. homomorphic encryption. homomorphism. computer security. data anonymization.

LIST OF FIGURES

LIST OF ACRONYMS

PKI	Public Key Infrastructure
HES	Homomorphic Encryption Standard
PHE	Partially Homomorphic Encryption
SHE	Somewhat Homomorphic Encryption
LFHE	Leveled Fully Homomorphic Encryption
FHE	Fully Homomorphic Encryption
AGCD	Approximate Greatest Common Divisor
GCD	Greatest Common Divisor
LWE	Learning With Errors
RLWE	Ring Learning With Errors
NTRU	Number Theory Research Unit
NIST	National Institute of Standards and Technology
SVP	Shortest Vector Problem
CVP	Closest Vector Problem
MPC	Multi-Party Computation
PIR	Private Information Retrieval

CONTENTS

1 INTRODUCTION

Technology is constantly present in people's lives, making it necessary to provide several personal information to computational systems, which will process this data, analyze and execute operations on it. With personal data processing, comes the need for privacy-preserving solutions. The problem is the lack of trust in the processing of this knowledge by third-party applications. Several times newspapers around the world show countless examples of exposition of confidential information. This information is usually transmitted using encryption such as TLS, to guarantee data confidentiality. However in order to analyze this data (in machine learning, for instance), it is necessary to operate it on clear text, needing to decrypt it first. At this moment, the data is vulnerable and susceptible to attacks or leaks of private information (by the cloud server), as will be explained hereafter.

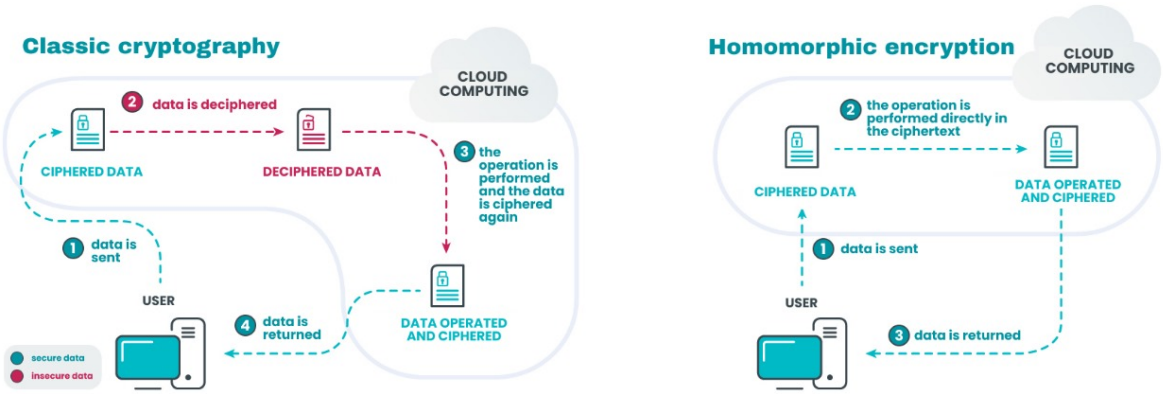
To exemplify how homomorphic encryption could bring security benefits over private user data, we can cite the processing of information by cloud computing. In a scenario where there is no homomorphic encryption, the application receives encrypted data, decrypts it, and then performs over clear text. At this moment, the information is unprotected, being subject to attacks or even, malicious use of the data by the server that is doing the analysis. After the data is processed, the result is encrypted and sent back to the user. Now, if we use homomorphic encryption, there would be no need to decrypt the data at any moment in the flow of the data analysis, since the operations of this type of scheme can be made directly on the encrypted text, letting the data protected and safe all along the way. Figure 1 shows the difference between the two possibilities.

Homomorphic encryption has the potential to solve many security and privacy issues, in a variety of daily applications, such as cloud computing, data science, and zero-knowledge proof, among others (ALAGIC et al., 2017). Besides that, it can help applications to operate in conformity with data protection laws, such as the Brazilian LGPD (Lei Geral de Proteção dos Dados) or the European GDPR (General Data Protection Regulation), because user data will not be decrypted.

So, we can observe there are several benefits in using homomorphic encryption, nevertheless, it is important to highlight that there are some limitations on the operations that can be made on encrypted data (as addition and/or multiplication) and the number of times it can execute on (depth). Therefore, different schemes have different properties. There are a few classifications of homomorphic encryption:

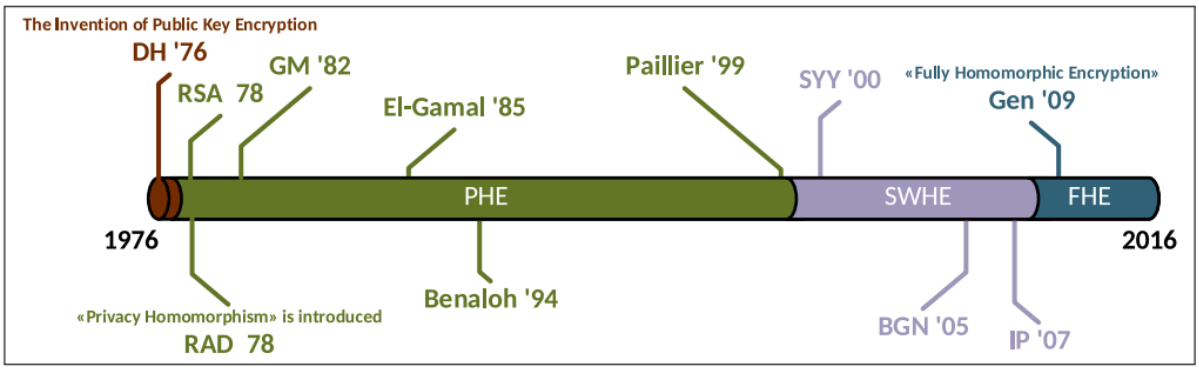
- Partially Homomorphic Encryption (PHE): supports only one type of operation, such as addition or multiplication. Does not have a restriction on the number of operations that can be computed;
- Somewhat Homomorphic Encryption (SHE): supports two types of operation, but can be operated only for a specific number of times (the most common case is an unlimited number of additions and a small number, usually one, of multiplications);

Figure 1 – Flow of information exchanged between servers using and not using homomorphic encryption



Source: Translated from (KUNDRO, 2019)

Figure 2 – Homomorphic schemes and its variants over the years



Source: (ACAR et al., 2018)

- Leveled Fully Homomorphic Encryption (LFHE): allows any type of operation (addition and multiplication) for a pre-established depth (the specific number of times it allows to be made);
- Fully Homomorphic Encryption (FHE): allows any type of operation with unlimited depth. This is the most difficult to achieve mathematically and, also, the one with more applications.

In Figure 2, it is shown the timeline of homomorphic encryption over the years, since the invention of the public key infrastructure (PKI) by Diffie-Hellman in 1976 (DIFFIE, 1976). In this info graph, we see some traditional schemes like RSA (RIVEST; SHAMIR; ADLEMAN, 1978) (multiplicative homomorphic), Goldwasser-Micali (GOLDWASSER; MICALI, 1982) (homomorphic over addition over binary numbers) and ElGamal (ELGAMAL, 1985) (multiplicative homomorphic and, according to (KNIRSCH et al., 2020), there is a variant where it is additive homomorphic). These schemes contain the properties of PHE. Only in 2009 it was created the first FHE scheme, by Gentry (GENTRY, 2009), having several improvements in performance and security ever since.

The concept of homomorphic encryption is recent, as well as its applications. It is also important to say there is an effort to standardize homomorphic encryption (ALBRECHT et al., 2018), the so-called Homomorphic Encryption Standard (HES). At this moment, the standard has only some specifications of operations that a homomorphic encryption scheme must have, to be considered as such, some secure parameters for well-known FHE schemes, such as BGV (BRAKERSKI; GENTRY; VAIKUNTANATHAN, 2014) and BFV (FAN; VERCAUTEREN, 2012). These schemes can be transformed into FHE using Gentry's proposal (GENTRY, 2009; ALBRECHT et al., 2018). Both schemes are based on the Learning With Errors (LWE) or Ring Learning With Errors (RLWE) problems, which will be explained in more detail later. The standard also mentions some new approaches and some properties that future schemes must contain, to make it more secure or functional.

1.1 GOALS

1.1.1 General goals

Our main objective is to perform a theoretical study of homomorphic encryption, identifying the underlying security assumption of a variety of known schemes and possible applications in several scenarios. Particularly, we focus on the study of PHE and SHE, and only briefly discuss LFHE and FHE.

1.1.2 Specific goals

- Introduce the fundamental concepts of PHE and SHE schemes, as well as their theoretical foundations;
- Explain the specifications of homomorphic encryption schemes, focusing on PHE and SHE;
- Show applications for all variants;
- Implement a PHE scheme from those studied and presented;
- Implement a practical application for the selected PHE scheme.

2 MAIN CONCEPTS

In this chapter, we will present the necessary background to study homomorphic encryption, as well as some important concepts necessary to understand specific HE schemes.

2.1 CRYPTOGRAPHY

The humankind has developed some very secure mechanisms to transmit data safely, which means, keeping its integrity and confidentiality, from one point to another. Whereas wars are horrible in so many ways, it is also a stage for the creation of multiple technologies throughout history. Some more recent examples of such inventions in difficult times, over World War II are the vaccines, jet engines, radar, and electronic computers (which were created, initially, to break the German ciphering machine, the Enigma) (LITTLE, 2021).

One of the first encryption schemes proposed in history is the Caesar cipher, used in ancient Rome, more than two thousand years ago, created by Julius Caesar himself. Because of the war, he needed a way of communicating with his army with the intention that, if the message could have been compromised somehow (obtained by the enemy, for example), the message would still remain secure and only his army would be able to decrypt the real message.

The Caesar cipher is based on a very simple technique, in which the letters of the alphabet are skipped a certain number of times. For example, if in the original message (usually called clear text) is used the letter "A" and the "secret" to decrypt it is "plus 3", then we would replace all the "A's" in the message with the letter D. With an alphabet of 26 letters, we could say that the ciphering C of a clear text m is given by the formula: $C \equiv m+3 \pmod 2)6$.

The difficulty of finding the solution to this cipher technique is not so hard, because a simple brute force algorithm that tries all 26 possibilities of shifts can decrypt the whole ciphered message. Another interesting fact to notice about this cipher method is that the same secret key is used to encrypt and decrypt it, which is called symmetric key encryption. That happens because we encrypt using the idea of shifting n letters in the alphabet and, to decrypt we also use this idea, but reversed.

As the years passed, humankind evolved its encryption schemes. Several ideas have been used and improved over the years, using alphabetic substitution, transposition, rotor machines, and one-time pad techniques. It is also important to say that our technology to create such schemes has also evolved, enabling us to create schemes that are much more secure, using more sophisticated techniques. The computers and electromechanical machines made that happen, as well as our development in mathematics.

Although symmetric key encryption (also called private key encryption, because everything relies on one secret key) is very effective, there is one major problem: the key distribution. To communicate with one another in a secure way using a symmetric key, both parties need to previously agree on the same secret key. But how to deliver this secret key over an insecure channel? In order to solve that, public key cryptography was proposed.

In 1976, two mathematicians developed a new way of cryptography, changing the modern technology world. In the official introduction to their theory, Diffie and Hellman (DIFFIE, 1976) presented asymmetric encryption, also called public key cryptography, which consists mainly of two types of keys: the public key (which is shared with everyone) and the private one (only the owner of the key is supposed to have access).

The way this mechanism works is: anyone can encrypt some clear text using someone's public key and only the private key owner can decrypt it. In other words, all data encrypted with one of the keys can be decrypted with the other key. The only mechanism required to implement it is having one or many trusted servers, storing all the trusted public keys. It is very simple and effective, not needing to worry about private key distribution anymore. As the public key is public information, it should be infeasible to obtain any information about the private key from it. It needs to be constructed mathematically and computationally hard to do so. With public key cryptography, networks can communicate securely and effectively. Clients can trust servers and vice-versa based on this assumption. It then, ensured confidentiality, integrity, and even authentication, depending on how we use public key encryption.

On the Internet, both symmetric and asymmetric encryption are frequently used. One example is TLS, which aims to guarantee safe communication between client and server by exchanging all messages encrypted. In this application, both symmetric and asymmetric cryptography are used. The client who wants to connect to the server sends a request with a symmetric key in it (encrypted using the server's public key). The server accepts the connection and saves the symmetric key to send everything encrypted. This is used for both security and speed since symmetric cryptography is much faster than the asymmetric one.

One of the first public key encryption schemes, and the most used nowadays, is RSA. The scheme is based on the difficulty of finding any two prime numbers, that result in an arbitrary integer when multiplied, known as the factorization problem. Because of its importance in transmitting shared keys for symmetric-key cryptography (like TLS) and its homomorphic properties, we present next the algorithms of this famous scheme. Normally an encryption scheme consists of three algorithms: key generation, which generates the key to encrypt and decrypt; the encryption algorithm, which takes the message, the key, and outputs the ciphertext; and the decryption algorithm, which takes the ciphertext, the key (normally on the reverse way it is done on encryption algorithm) and outputs the clear text. We will show these algorithms, and also, the multiplicative homomorphic property.

Key generation Consider large prime numbers p, q, calculate $n = pq$ and $\phi = (p-1)(q-1)$. Take e as a small number, and its corresponding multiplicative inverse d such that $ed \equiv 1 \pmod{\phi}$. The public key is (e,n) and the private key is (d,n). Given n and e, it is computationally infeasible to factor it and obtain p and q, and consequently ϕ and d.

Encryption Define $\text{ENC}(m) = m^e \bmod n$, where m is the message to be encrypted, with $0 \leq m \leq n$.

Decryption We can retrieve the original message m, from the ciphertext c with the following: $\text{DEC}(c) = c^d \bmod n$.

Homomorphic property We stated that RSA is multiplicative homomorphic and we can show that. Multiplying two ciphertexts would generate the following:

$$\begin{aligned} \text{ENC}(m_1) \times \text{ENC}(m_2) &= (m_1^e \mod n) \times (m_2^e \mod n) \\ &= (m_1^e \times m_2^e) \bmod n \\ &= (m_1 \times m_2)^e \bmod n \\ &= \text{ENC}(m_1 \times m_2) \end{aligned} \tag{2.1}$$

With that, it can be observed that multiplying two ciphertexts is equivalent to raising the product of the plaintexts to the power of the secret key (Yackel, Ryan, 2021). This means that if we multiply two numbers that were encrypted using the same public key (of the same person) with the RSA scheme, it is equivalent to multiplying the two numbers and then encrypting them. More examples and properties of this definition will be provided in the next section.

2.2 HOMOMORPHIC ENCRYPTION

Homomorphism is, in mathematics, a mapping between two algebraic structures of the same type, that preserves the operations of these structures. Consider the function f defined as $A \to B$ (being A the function's domain and B the respective image), where A and B are sets equipped with a binary operation $*$. The homomorphism preserves the operation of the structures, meaning $f(a*b) = f(a)*f(b)$ for all elements $a, b \in A$. When we study homomorphic encryption, the function f is the encryption or decryption function, with the operation $*$ depending on the scheme.

Although public key cryptography is very efficient and solves many of the problems in today's network communication, there are several cases where public key encryption on its own is not enough. In all examples given, the data is encrypted throughout the whole flow, from the client to the server and vice-versa. But whenever is necessary to perform any search or operation in the data itself, it is required to decrypt it in the server and, if the server does not have access to the private key, it is not possible to do so. In other words, if the server needs to decrypt the data it needs to know the private key. More importantly, the computations performed over this possible confidential information are done without cryptography at all. This means that a malicious server could take that data and provide it to some other service or keep it on its private database, without the user's consent.

Nowadays with the need to expand and cheapen the services, several technology companies appeal to the famous Cloud computing resources. When a cloud server is used to handle data processing, such as in data mining and Big Data applications, the server needs to have access to the data. Even having some agreement between the cloud service engaged with

the user, the user is not completely aware of what the server is going to do with the information provided.

Even though distributed computing paradigms in general are very efficient and powerful, such as Cloud computing, fog computing, or edge computing, there are some security issues to be considered. When it comes to private and sensitive information, it is necessary to handle it very carefully and, in the best scenario, not have access to unencrypted data at all. As the information gets decrypted in some parts of the data flow, data breaches can occur. Applications that use sensitive information cannot fully trust a server to analyze their data, because of the aforementioned problem. Some of these applications are medical data processing, outsourcing of financial operations, statistics over sensitive data, online voting, multi-party computations, and machine learning (CHILLOTI, 2022; ROCHA; LÓPEZ; ROCHA, 2018).

In 1978, it was already foreseen by the people who invented the famous RSA cryptosystem (RIVEST; SHAMIR; ADLEMAN, 1978), that some class of schemes could act on the ciphered text without decrypting it. This class of schemes were called of homomorphic encryption. The problem is that many researchers and mathematicians tried to reach such a scheme and failed. It was found out that many earlier known and very common public key cryptosystems such as RSA (RIVEST; SHAMIR; ADLEMAN, 1978), ElGamal (ELGAMAL, 1985), and Paillier (PAILLIER, 1999) already had homomorphic properties over specific operations. However, it could only perform one type of operation over the ciphertext. This was named partially homomorphic encryption (PHE). For instance, both RSA and ElGamal have a homomorphism over multiplication and Paillier over addition.

So far we have seen RSA as an example of PHE, which accepts any number of multiplications. However, we also have somewhat homomorphic encryption (SHE), that supports two types of operations over the ciphertext, but only for a determined number of times; leveled fully homomorphic encryption (LFHE), that supports a set of operations for a pre-determined depth (by the user's choice); fully homomorphic encryption (FHE), that supports any operations for any depth. The last one was believed to be impossible, until 2009, when Gentry presented as his doctoral proposal a fully homomorphic encryption scheme (GENTRY, 2009). After that, the frontiers of homomorphic encryption were open, enabling it as a real use case.

The problem of making a scheme that supports a set of operations, for any number of times, relies on one parameter of probabilistic encryption schemes: the noise. It is necessary to add some noise into the encryption, so it remains "unique" and guarantees its security and correctness. The encryption is generally not the problem, but when we decrypt into some modular area, the noise becomes an issue, because it demands to be smaller than a certain value (a static parameter from the scheme), and homomorphic operations increase that noise. More details will be presented in the next subsection.

2.2.1 Noise

Noise is a parameter of encryption functions in probabilistic schemes, such as fully homomorphic schemes. It is necessary to apply some randomness to the encryption channel, so only the owner can remove it and obtain the original message. When we "include" some random parameter in cryptographic functions, they end up not being deterministic, but rather probabilistic. To that randomness, we call it error or noise. If the encryption of the same message is always the same ciphertext, then we call it a deterministic scheme. This can result in numerous attacks, as will be shown hereafter. In order to make these schemes more secure, we can add some randomness to them, so the ciphertext for the same message is always different.

Over the years researchers were able to find possible attacks over deterministic schemes, which depreciate the system security. In (HOVD, 2017), is explained how some attacks are performed over deterministic schemes like RSA, making it insecure due to its deterministic properties, and why noise is demanded in general encryption schemes. This happens, in part, because the encryption of the same message is always the same, which does not happen in probabilistic schemes.

Now that we know why noise is important to encryption schemes, the question is how this can disturb the decryption over operated ciphertexts. The problem is that "classic" encryption schemes do not change their ciphertext after it is encrypted, but homomorphic ciphertexts are changeable, because of the possible operations over it. To exemplify that, we will show a SHE scheme by Gentry and some other researchers in the field in 2010 (DIJK et al., 2010), called DGHV. This was a simple arrangement made by them to show an application of Gentry's doctoral proposal, the bootstrap approach, which shall be explained in the next subsection. Next, we show the main functions of the symmetric version of this cryptosystem, extracted from their work (DIJK et al., 2010), where only one bit is encrypted, which means our message $m \in \{0,1\}$.

- Key generation: the key p is an odd integer, chosen from some interval $p \in [2^{n-1}, 2^n)$, where n is a security parameter;
- Encrypt(p,m): to encrypt a bit $m \in \{0,1\}$, set the ciphertext as an integer whose residue mod p has the same parity as the plaintext. Namely, set $c = pq + 2r + m$, where the integers q, r are chosen at random in some other prescribed intervals, such that $2r$ is smaller than $p/2$ in absolute value;
- Decrypt(p,m): output $(c \bmod p) \bmod 2$.

It can be shown that this scheme is a SHE scheme, given the fact we can make additions and multiplications between two different ciphertexts c_1 and c_2 generated with the same private key p, and maintain the homomorphic encryption property over these operations. These statements can be proved in the equalities below, being the symbol $+$ the usual addition and the symbol $\times$ representing the usual multiplication:

$$\begin{aligned}
\text{ENC}(m_1)+\text{ENC}(m_2) &= (pq_1+2r_1+m_1)+(pq_2+2r_2+m_2)\\
&= p(q_1+q_2)+2(r_1+r_2)+(m_1+m_2)\\
&= pq_3+2r_3+(m_1+m_2)\\
&= \text{ENC}(m_1+m_2)\\
r_3 &= r_1+r_2\\
q_3 &= q_1+q_2
\end{aligned} \tag{2.2}$$

$$\begin{aligned}
\text{ENC}(m_1)\times\text{ENC}(m_2) &= (pq_1+2r_1+m_1)(pq_2+2r_2+m_2)\\
&= pq_4+2r_4+(m_1\times m_2)\\
&= \text{ENC}(m_1\times m_2)\\
r_4 &= 2r_1r_2+r_1m_2+r_2m_1\\
q_4 &= pq_1q_2+q_2m_1+q_1m_2
\end{aligned} \tag{2.3}$$

Here some mathematics were hidden to keep the general understanding, but what is important to note is that the noise from addition (r_3) or multiplication (r_4) needs to be $< p/2$, so decryption works. But taking the addition first, we have r_1 and r_2 that are very distant from $p/2$, which would require many additions of ciphertext before $2r_3$ reaches $p/2$. This does not happen with multiplication, because r_1 and r_2 are multiplied by all the other factors, growing much faster, so r_4 becomes larger than $p/2$ much faster.

This means that, at each operation made into the ciphertext, it will make the noise (r_i) to grow a little more. If the noise becomes somehow larger than the secret key p, the decryption will be incorrect. This happens because we perform a modular operation to decrypt a ciphertext, making the modular area to be incorrect, as shown below for a ciphertext c:

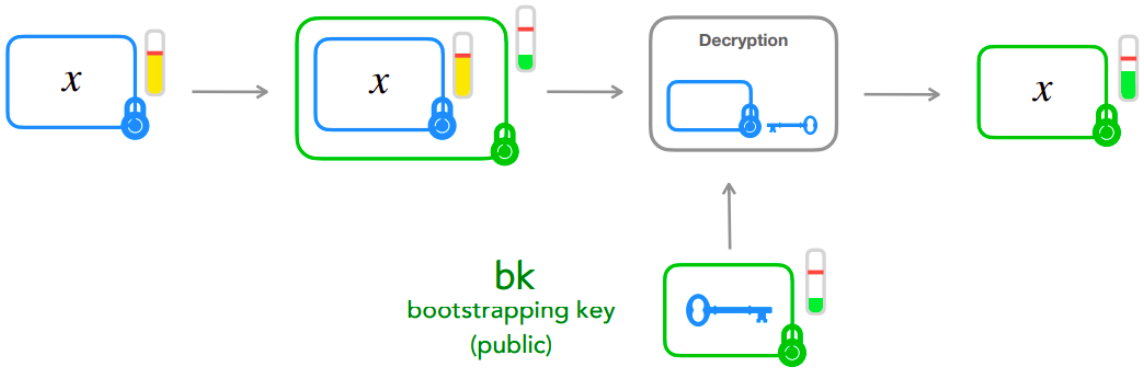
$$\begin{aligned}
\text{DEC}(c) &= (c \bmod p) \bmod 2\\
&= ((pq+2r+m) \bmod p) \bmod 2
\end{aligned} \tag{2.4}$$

Because of that, this scheme is restricted to only a few operations over the ciphertext. This is the reason why the scheme is considered a SHE. When the noise becomes larger than the secret key, it is said the scheme becomes unmanageable. Otherwise, Gentry found a way to surpass such problems, enabling almost all SHE schemes to become FHE (GENTRY, 2009), which means, being able to make any amount of operations over the ciphertext, using what he calls bootstrap technique, which will be presented next.

2.2.2 Bootstrap

As an old open problem until 2009, Gentry's bootstrap proposal was a mark in the world of cryptography. Besides creating an FHE, he also created a method to convert a SHE

Figure 3 – Bootstrap representation



Source: (CHILLOTI, 2022)

scheme into an FHE one. As we noticed from the previous subsection, the difficult problem in making an FHE scheme is how to manage the noise growth, so, his approach is based on the idea of shrinking the noise when it gets too large. To demonstrate that, first, we need to provide some information about the whole process. To bootstrap some ciphertext, we need two new elements in the encryption scheme, a public evaluation key (also called bootstrapping key) and a *recrypt* function. The evaluation key is used in the recryption process, or in the relinearization process (it will be explained later in this work). When the key is only used in the relinearization process, we call it to be a relinearization key.

The recrypt function is used to reduce the noise when it gets too large. In order to do that, it is necessary to re-encrypt the ciphertext with an evaluation key k, and this will reset the noise. This process gives us a ciphertext of a ciphertext. To obtain the ciphertext of the original message x, we use the evaluation key again, which is a public value. Let us take the expression $c = \text{ENC}(x,k)$ as the original ciphertext and $c' = \text{ENC}(c,k')$ as the encryption of the first ciphertext. Using the evaluation key, we can decrypt c' and get the ciphertext of the original clear text. More details are given in Figure 3, where we present an example of how the bootstrap procedure is applied.

Taking an SHE scheme as presented previously, like DGHV we showed before as an example, where is only possible to arrange some operations over the ciphertext because of its noise growth, we show how the bootstrap works based on the analogy of Figure 3. Imagine the last operation possible before exceeding the noise limit. At that moment it is not possible to make any other homomorphic operation on the ciphertext, otherwise, the decryption will fail. That is represented as the blue box, and the yellow bar next to it represents the noise growth, and the red line is the limit of its growth, or it will get unmanageable.

The second step is the recrypt function, here represented as the green box. As can be seen, the noise of the new encryption is reset as it is a new encryption. At this moment, we have the encryption of the ciphertext of the original clear text. To get the encryption of the clear text, we need to use the evaluation key to decrypt only what is inside the green box, which means,

decrypt only the blue box homomorphically. The operation of decrypting the blue box inside the green box adds a small amount of noise, but now we have the encryption of the original clear text inside our green box, with a smaller noise. This makes it possible to make some new operations over the ciphertext and re-do this process as many times as necessary.

The biggest problem with bootstrapping is how expensive it is. Every time the noise is about to get unmanageable we need to bootstrap (recrypt) it, and we need to make some adjustments in the ciphertext before doing that. Imagine we want to perform one thousand multiplications over an SHE scheme, that allows one multiplication before it gets unmanageable. It would be necessary at each multiplication, to bootstrap it (DUCAS; MICCIANCIO, 2015; CHILLOTTI et al., 2020; MARCOLLA et al., 2022). Some new research points out some enhancements to this method, making it possible to use in real scenarios, otherwise, it would be infeasible. This new line of research into homomorphic encryption is called fast bootstrapping, which has the goal of making the recryption function faster, as well as all the operations involved in it.

Despite all the efforts to make efficient FHE schemes, the sizes of the public and evaluation keys are still too big and its operations and decryptions take too long. Therefore, there is also another branch of the FHE, the LFHE, that competes with the bootstrap approach. Some examples of LFHE involve the schemes FHEW (DUCAS; MICCIANCIO, 2015), TFHE (CHILLOTTI et al., 2020), and CKKS (CHEON et al., 2017), the last being the fastest and most popular.

2.2.3 Leveled approach

LFHE schemes can be very often related to more practical uses nowadays than the bootstrap approach, due to its speed in performing operations over the ciphertext and decrypting it. This happens because LFHE can only make a limited amount of operations, which the user can specify. If we are aware of the amount of times we need to operate on the ciphertext, we can make some improvements so the noise gets controlled. We need more mathematical background to make the enhancements necessary to keep the noise contained. This is especially essential when it involves costly operations, such as multiplications. Some schemes like CKKS involve up to three additional steps in the process of multiplying two ciphertexts, for instance.

Because of the careful noise management, this HE variant is also related to different public key sizes. Actually, its size is dependent on the depth the scheme must accept. Therefore its operations speed is also related to that, which makes this type of scheme very useful when we are aware of the number of operations we need over the ciphertext, and this amount is not very large. Otherwise, we still need to use the bootstrapping technique. Further information will be presented in the section regarding LFHE Section 5.1, but details on the schemes' implementation will not be presented, as it is out of the scope of this work.

2.3 DIFFERENT CLASSES OF SCHEMES

There are several ways we can achieve a fast and secure encryption scheme. One of the most popular nowadays is lattice schemes, especially for its resistance to quantum computers. As we cited in Section 2.1, RSA is derived from the factorization problem, which is computationally difficult to solve when we do not have access to private information. Although this can be difficult for today's computers, with the arrival of quantum computers, this is about to change. As this device can perform exponential calculations, creating a superposition of these, a powerful quantum computer would be able to break RSA encryption in a couple of minutes, using the famous Shor's algorithm (SHOR, 1994).

After that demonstration, mathematicians decided to explore solutions that would be resistant to quantum computers, known as post-quantum cryptography. The National Institute of Standards and Technology (NIST), which is responsible for finding good practices on cryptography schemes and other initiatives in the United States of America, has been making campaigns to search for the best post-quantum schemes. Considering all submissions to this institute, the great majority have used lattice-based schemes, proving to be one very versatile line of schemes based on hard problems. A hard problem is also said to be NP-hard, which means, in short, that a computer is not able to find a solution in polynomial time, with any tools known nowadays. It has also been proved that this class of algorithms is average-case hard (LANGLOIS; STEHLÉ, 2015) adapting some parameters.

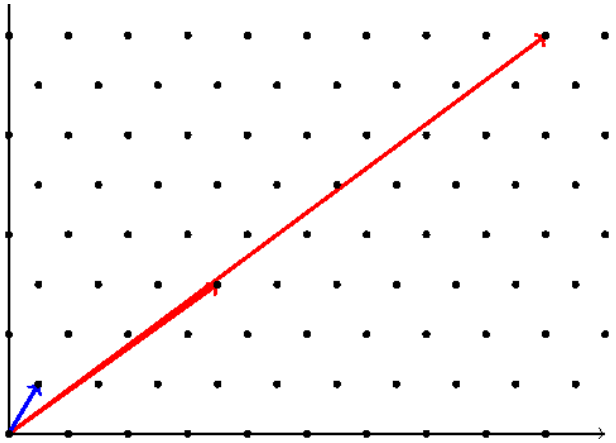
In cryptography, we have many different types of underlying problems we can use to ensure the keys remain secure. Some mathematical problems we can arrange to create cryptosystems such as factorization, discrete logarithms, LWE (Learning With Errors), hash functions, code-correcting, multivariate functions, and lattice. Some types of mathematical problems will be defined in the next subsections. We need to be aware that lattice problems are very important, because as exposed by Gentry and in some surveys about homomorphic encryption (MARTINS; SOUSA; MARIANO, 2017), the most important and reliable at the moment is lattice-based, because of its post-quantum properties and arrangements that make homomorphic encryption easier.

2.3.1 Lattice

Firstly, we need to define a lattice. According to (ALWEN, 2020), the main definitions are as follow:

> Lattice: A lattice can be thought of as any regularly spaced grid of points stretching out to infinity.
>
> Vector: A vector is a tuple of points called the coordinates of the vector. So (2,3) is a particular 2-dimensional vector as it has 2 coordinates, and a lattice is a collection of evenly spaced vectors. A special vector of interest is the origin which is the vector with

Figure 4 – SVP demonstration, where the blue arrow is the shortest vector to the origin and the red arrows are other possible vectors



Source: (BRIGHT, 2013)

all coordinates set to 0. For example, in 3 dimensions, the origin is (0, 0, 0). We say that a vector is long if it is far away from the origin. Conversely, a vector is short if it is close to the origin.

Basis: Lattices are infinitely large objects but computers only have a finite amount of memory to work with. So we will need a succinct way to represent lattices if we are going to use them in cryptography. For this, we use what is called a basis of a lattice. A basis is a small collection of vectors that can be used to reproduce any point in the grid that forms the lattice.

So we can say that a basis can form any vector from the lattice, which is very important in lattice schemes. We can also call short vectors the ones whose lengths between one another are relatively small compared to other vectors in the lattice. Long vectors are the opposite, being long between one vector to another. A short basis is a set of short vectors. A long basis is a set of long vectors. Now we can specify the two most important problems in lattice-based schemes, the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP).

The SVP is based on the difficulty of finding the closest vector to the origin, given a long basis for a lattice. The problem here is the complexity of managing basis in the order of 10 thousand vectors. The CVP lies in the difficulty of finding the closest vector to an arbitrary point P in a long basis lattice. One illustrated example of the SVP and, as well as the CVP with $P = 0$ can be viewed in Figure 4. Taking the origin in Figure 4 with some arbitrary basis, it is shown some points of that basis. The SVP problem is based on the difficulty of finding the shortest vector to the origin, here represented as the blue vector (blue arrow). Whereas it is easy to be seen that way, it is not easy to compute it when we have a very large basis and several dimensions.

In order to get clearer on how to read the representations of lattices and their mathematics in the next chapters, we will show the representation and notation for lattices in general, using (KAWACHI; TANAKA; XAGAWA, 2007) as a base. In this way, the length of a vector

$x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is denoted by $\| x \| = (\Sigma_{i=1}^{n} x_i^2)^{1/2}$. The inner product of two vectors x and y is denoted by $\langle x, y \rangle$, which is $\Sigma_{i=1}^{n} x_i \times y_i$. The dimension of a lattice is, normally, taken as the parameter n. Regarding the closest vector, we also have a representation for that, being $\lfloor x \rceil$ the closest integer to $x \in \mathbb{R}$ and $frc(x) = |x - \lfloor x \rceil|$, being the distance from x to the closest integer.

We can define a n-dimensional lattice mathematically in $\mathbb{R}^n$ as the set $L(b_1, \ldots, b_n) = \Sigma_{i=1}^{n} \alpha_i \times b_i$, $\alpha_i \in \mathbb{Z}$, of all integral combinations of n linearly independent vectors $b_1, \ldots, b_n$, which are called as the *basis* of lattice L. So, a basis can be represented as a matrix $B = (b_1, \ldots, b_n) \in \mathbb{R}^{n \times n}$ whose columns are the basis vectors. We are only using the matrix dimension $n \times n$, which is also called full-rank lattices, because it is what we are interested in for the problems presented in this work, however, there are other constructions possible.

We say that two bases are equivalent if they span to the same lattice, i.e., they produce the same lattice. Two bases B, B' can be considered equivalent, if and only if there exists a matrix $U \in \mathbb{Z}^{n \times n}$, with $det(U) = \pm 1$, such that $B' = BU$. If any point from a lattice $L(B')$ belongs also to another lattice $L(B)$, we say that $L(B') \subseteq L(B)$ and also that $L(B')$ is a sublattice of $L(B)$. We can consider the *fundamental parallelepiped* $\rho(B)$ as $\rho(B) = \{\Sigma_{i=0}^{n} a_i \times b_i \,|\, a_i \in \mathbb{R}, a_i \in [0,1)\}$. With this construction, we can "embrace" the whole lattice with the fundamental parallelepiped. Another important property is the volume of $\rho(B) = |det(B)| = det(L)$. Some further definitions may be presented, as specific properties for each scheme.

2.3.2 AGCD

The Approximate Greatest Common Divisor (AGCD), also called the Approximate Common Divisor (ACD) problem is based on the difficulty of determining a secret integer p when it is given many fragments of the form $\text{ENC}(c_i) = pq_i + e_i$, where the ciphertext is c_i, the noise is e_i, and q_i is some multiplication factor. These parameters are lattices with a very large basis, in other words, a very large lattice.

It has been widely used in the HE since the first publication of the DGHV scheme (DIJK et al., 2010), which was already described previously in this work, having many variations of that and many other schemes based on that idea. This is done by the difficulty in finding the Greatest Common Divisor (GCD), which means, the positive integer that divides two numbers without a remainder in very large lattices. In HE, it is the most used alongside LWE schemes, because of its simplicity and considerably fast execution to either encrypt or decrypt.

2.3.3 LWE

In 2005 a probabilistic encryption scheme was built by Regev (REGEV, 2010), called Learning with Errors (LWE). The basic approach is the difficulty in retrieving the secret key s and the noise e from the equation: $t = gs + e$, where g is a given sequence of random numbers and t is the public key, also being a sequence of numbers.

To explain this method in more detail, we provide an example where we can only encrypt bit numbers, based on the example given in (BUCHANAN, 2023). For that, it is necessary to generate a secret key s and a noise e. For the purpose of this example, we choose $s = 5$, $e = 12$. Now we will generate a sequence of 10 random numbers, the $g = [5, 8, 12, 16, 2, 6, 11, 3, 7, 10]$. Now we have all the values in the formula to build our public key t, generating $[37, 52, 72, 92, 22, 42, 67, 27, 47, 62]$. To encrypt, we use the function: $c = \sum_{n=1}^{s} rand(t, s) + m$, $m \in 0, 1$ is the message to encrypt, and *rand* the function taking randomly 5 elements from the public key, since $s = 5$ and we have $rand(t, s)$. To decrypt the ciphertext and get the original message, we take $m \equiv c \pmod{s}$. The decryption is successful because the error is small, and t is very large. In this way, the noise added does not interfere, as it is very small, but it makes it computationally hard for an attacker.

Here we only show a toy example with integers, but we can construct encryption schemes using matrices or lattices instead. It has been considered to be quantum-resistant and can be combined with some other mechanisms. The Ring Learning With Errors (RLWE), for instance, is one of the most important, which encapsulates the simple idea of LWE into mathematical rings.

It is important to point out that the majority of FHE or LFHE schemes have been using RLWE approaches to reach fast and secure schemes, being the most prominent ones: CKKS, TFHE, BFV, and BGV. Also, there is some research on converting LWE schemes into AGCD ones (CHEON; STEHLÉ, 2015), which can make the process of homomorphic operations faster. Some more details about these schemes will be provided further in this work.

2.3.4 NTRU

NTRU is a public key cryptography system that allows not only encryption but also digital signatures. It is one viable and very efficient option to overthrow quantum computers as well, besides being five times as fast as RSA to encrypt and twice as fast to decrypt (HARJITO et al., 2022). Its whole structure is based on the SVP in lattices combined with the difficulty of factoring polynomials in a truncated polynomial ring.

This class of schemes has been a finalist in the post-quantum NIST standardization process. In addition to that, it has been the only class that was able to provide multi-key encryption (LÓPEZ-ALT; TROMER; VAIKUNTANATHAN, 2012), meaning two or more people can encrypt the same message, so only they are able to decrypt it. This is very important in Multi-Party Computations (MPC), in which multiple parties make calculations using their combined data. Some possible MPC FHE applications combine anonymous data from medical centers or blockchain usages.

NTRU is a class of schemes that is in constant development, with many variants and security proofs, not only against quantum computers but also against classical cyber security attacks. Despite all its usages, in HE we do not have many examples of schemes based on NTRU. The first HE scheme based in NTRU by Lopez-Alt-Tromer-Vaikuntanathan (LÓPEZ-

ALT; TROMER; VAIKUNTANATHAN, 2012), where it is possible to encrypt and perform computations using multiple private keys. Over the years, some researchers were able to apply some attacks over FHE NTRU-based schemes, as we can check on (ALBRECHT; BAI; DUCAS, 2016; CHEON; JEONG; LEE, 2016; KIRCHNER; FOUQUE, 2017), because of its overstretched parameters of modular rings. Another problem they mentioned was the evaluation key for bootstrapping it, which was leaking details of its private key.

Over the last years, new attempts to make an FHE scheme based on NTRU security were performed as proposed in (MITTAL; RAMKUMAR, 2022). The attacks were solved, by decreasing the size on the modulo Ring, changing the NTRU characteristic (using matrices, fields, or algebra), and removing the evaluation key as in (DORÖZ; SUNAR, 2020). The multiplications performed were faster, and no attacks have been performed over the new schemes so far, which can be still considered a good approach for FHE schemes. The HES promised to look at its security in the next round of meetings over the subject.

3 PARTIALLY HOMOMORPHIC ENCRYPTION

Partially homomorphic encryption are encryption schemes that allows only one type of operation over encrypted data. In other words, we can apply one operation as many times as needed into the ciphertext, and we will still be able to decrypt into the corresponding clear text normally. It is important to notice that probabilistic schemes of this variant of HE also need to handle noise, otherwise, the decryption can fail as explained (see Subsection 2.2.1), but the structures are much simpler. As this is the simplest type of homomorphic variant, a brief explanation of the properties of each scheme will be given, as well as the underlying problem. We will explain the key generation, encryption, and decryption procedure for each PHE scheme available nowadays, briefly showing their security assumptions, and describing the corresponding homomorphic operation.

Table 1 summarizes each scheme and its respective properties. It is also important to know whether a scheme is additive and/or multiplicative homomorphic by a scalar number, since some applications may require it. It is worth mentioning that some schemes are only homomorphic in theory, as the implementation must embrace many more layers of security and, usually when we do that, we lose the homomorphic property. This is the case of RSA, presented in Section 2.1.

3.1 ENCRYPTION SCHEMES

In this section, we present the main PHE schemes, denoting all the procedures necessary to encrypt and decrypt some data.

Table 1 – Summary of PHE schemes

Scheme	**Underlying problem**	**Op**	$\times K$	$+K$	^ K	**Space**
RSA	Factorization	$\times$				$\mathbb{Z}_n$
Goldwasser-Micali	Quadratic Residuosity Problem	XOR	✓	✓		$\mathbb{Z}_2$
ElGamal classic	Discrete Logarithm	$\times$			✓	$\mathbb{Z}_p$
ElGamal additive	Discrete Logarithm	+	✓			g^m
ElGamal Elliptic Curve	Elliptic Curve Discrete Logarithm	+	✓			
Benaloh	Residuosity problem	+	✓	✓		$\mathbb{Z}_n$
Paillier	Decisional Composite Residuosity	+	✓	✓		$\mathbb{Z}_{n^2}$
Naccache-Stern	Higher Residuosity	+	✓	✓		$\mathbb{Z}_n$
Okamoto-Uchiyama	Factorization + p-Subgroup	+	✓	✓		$\mathbb{Z}_n$
Damgard-Jurik	Decisional Composite Residuosity	+	✓	✓		$\mathbb{Z}_{n^{s+1}}$
Kawachi-Tanaka-Xagawa	Lattice (SVP)	+				$\mathbb{Z}_p$

The "Op" column stands for which operation the scheme is homomorphic by. *operation* K means if a scheme supports such operation between ciphertext and scalar. The space "column" means the ciphertext space, where n, p, g are specific parameters from each scheme.

3.1.1 RSA

The RSA scheme (RIVEST; SHAMIR; ADLEMAN, 1978) was already introduced in Section 2.1. We recall that RSA is a multiplicative homomorphic scheme, and until today, it is the most used cryptosystem.

3.1.2 Goldwasser-Micali

Goldwasser-Micali (GOLDWASSER; MICALI, 1982) is the first probabilistic public key encryption scheme. Although it has not been actually used in practice due to its large ciphertexts, many other schemes are derived from it. The security is based on the difficulty of retrieving p and q, of $n = pq$, given the Quadratic Residuosity. A number a is called *quadratic residue modulo n*, if there exists an integer x such that $x^2 \equiv a \pmod{n}$ citeacar2018survey, using the notation QR in the equation $(QR[n])$. The scheme is additive homomorphic for bits, which means we can operate using XOR. Another homomorphic property relies on the addition and multiplication over a scalar.

Key generation We compute $n = pq$, where p and q are primes. Choose an integer x, such that $\frac{x}{n} = \frac{x}{p} \times \frac{x}{q} = (-1) \times (-1) = 1$, being this ratio the $QR[n]$. The public key is (x,n) and the private key is (p,q).

Encryption The message is converted into a string of bits $(m_1, m_2, \ldots, m_k)$. For every m_i, a y_i is produced as $\gcd(y_i, n) = 1$. The encryption is performed as follows, for each bit m_i and y_i: $\text{ENC}(m_i) = y_i^2 \times x^{m_i} \pmod{n}$.

Decryption Consider c as the ciphertext. We need to decide if each c_i is a quadratic residue modulo n, or not. If it is, then the corresponding $m_i = 0$, otherwise $m_i = 1$. The concatenation of each m_i is the original clear text.

Homomorphic property Consider two on-bit messages m_1 and m_2. We can state the additive property (modulo 2) by multiplying the ciphertexts, as follows:

$$\begin{aligned} \text{ENC}(m_1) \times \text{ENC}(m_2) &= (y_1^2 \times x^{m_1} \pmod{n}) \times (y_2^2 \times x^{m_2} \pmod{n}) \\ &= (y_1 \times y_2)^2 \times x^{m_1+m_2} \pmod{n} \\ &= \text{ENC}(m_1 + m_2) \end{aligned} \tag{3.1}$$

3.1.3 ElGamal

ElGamal (ELGAMAL, 1985) is a very popular scheme, whose underlying problem is based on the discrete logarithm assumption. This means the hardness of finding x in $g^x = h$, where g, h are given elements of a finite cyclic group. The scheme is multiplicative homomorphic, and it is also possible to make calculations power a scalar, due to its power in g^x.

Key generation For two large prime numbers p and q, such that $q|(p-1)$. Take a cyclic subgroup G_q of $\mathbb{Z}_p^*$ of order q and generator g. Compute $g \equiv y^{(p-1)/q} \pmod{p}$. Select a random $x \in \mathbb{Z}_q$ and set $h = g^x \pmod{p}$. The public key is (p,q,g,h) and the private key is (x).

Encryption Generate a random number $r \in \mathbb{Z}_q$. As the output of the encryption function, a pair of ciphertexts will be made, as $\text{ENC}(m) \rightarrow (c_1, c_2)$. The encryption is calculated as:

$$\begin{aligned} \text{ENC}(m) &= (c_1, c_2) \\ c_1 &= g^r \pmod{p} \\ c_2 &= m \times h^r \pmod{p} \end{aligned} \tag{3.2}$$

Decryption In order to decrypt the pair (c_1, c_2) we need an auxiliary s, which is calculated as $s = c_1^x$. Then, the decryption occur as: $\text{DEC}(c1, c2) = c_2 \times s^{-1} \equiv h^{-r} \times (m \times h^r) \equiv m \pmod{p}$, where s^{-1} is the inverse of s in the group G_q.

Homomorphic property Let $\text{ENC}(m_1) = (c_1, c_2)$ and $\text{ENC}(m_2) = (c_1', c_2')$. The property will be shown by multiplying c_1 with c_1' and c_2 with c_2':

$$\begin{aligned} c_1 \times c_1' &= g^r \times g^{r'} \pmod{p} \\ &= g^{r+r'} \pmod{p} \\ c_2 \times c_2' &= (m_1 \times h^r) \times (m_2 \times h^{r'}) \pmod{p} \\ &= m_1 \times m_2 \times h^{r+r'} \pmod{p} \\ \text{ENC}(m_1) \times \text{ENC}(m_2) &= (g^{r+r'}, m_1 \times m_2 \times h^{r+r'}) \pmod{p} \\ &= \text{ENC}(m_1 \times m_2) \end{aligned} \tag{3.3}$$

3.1.3.1 Additive variant

As a variant of the classic ElGamal scheme, it is possible to make the same classic scheme to be PHE regarding addition (CRAMER; GENNARO; SCHOENMAKERS, 1997).

Key generation The same steps in the classic scheme can be followed, with the same public and private keys.

Encryption It also outputs a pair (c_1, c_2), but here we need to transform the message m which initially belongs to $\mathbb{Z}_q$, into a group element of g^m. Generically, for encryption, we only substitute the parameter m for g^m, as follows:

$$\begin{aligned} \text{ENC}(m) &= (c_1, c_2) \\ c_1 &= g^r \pmod{p} \\ c_2 &= g^m \times h^r \pmod{p} \end{aligned} \tag{3.4}$$

Decryption The process is the same as for classic ElGamal, substituting m for g^m in the original equation, with $\text{DEC}(c_1, c_2) = c_2 \times s^{-1} \equiv h^{-r} \times (g^m \times h^r) \equiv g^m \pmod{p}$.

Homomorphic property The multiplication of the ciphertexts now becomes the sum of the clear texts, as we can see through the next expressions:

$$\begin{aligned} c_1 \times c_1' &= g^r \times g^{r'} \pmod{p} \\ &= g^{r+r'} \pmod{p} \\ c_2 \times c_2' &= (g^{m_1} \times h^r) \times (g^{m_2} \times h^{r'}) \bmod p \\ &= g^{m_1} \times g^{m_2} \times h^{r+r'} \pmod{p} \\ &= g^{m_1+m_2} \times h^{r+r'} \pmod{p} \\ \text{ENC}(m_1) \times \text{ENC}(m_2) &= (g^{r+r'}, g^{m_1+m_2} \times h^{r+r'}) \pmod{p} \\ &= \text{ENC}(m_1 + m_2) \end{aligned} \tag{3.5}$$

3.1.3.2 Elliptic curve variant

Almost the same procedures as classical ElGamal can be performed using Elliptic curves over a finite field (KOBLITZ, 1994). It is analog to the additive homomorphic variant and considers the group elements as points in the elliptic curve, multiplication is replaced by point addition, whereas exponentiation is replaced by point multiplication. The following procedures are presented in (KOÇ; ÖZDEMIR; ÖZGER, 2021).

Key generation For a large prime p, choose an elliptic curve $\mathbb{E}(\mathbb{Z}_p)$ over finite field of p elements, such that the order of $\mathbb{E}(\mathbb{Z}_q)$ is divisible by prime q. Similarly, we choose a generating point P, select a random $d \in \mathbb{Z}_n$, and set $Q = [d] \times P$, where the operation $\times$ is point multiplication in $\mathbb{E}$. The public key is (P, Q, p, q) and the private key is (d).

Encryption Generate a random number $r \in \mathbb{Z}_q$ and compute the pair (R_1, R_2), which are points on the given curve. The $\oplus$ is the correspondent point addition on the curve.

$$\begin{aligned} \text{ENC}(m) &= (R_1, R_2) \\ R_1 &= [r]P \\ R_2 &= [m]P \oplus [r]Q \end{aligned} \tag{3.6}$$

Decryption In order to decrypt the pair (R_1, R_2), we need an auxiliary point S, which can be calculated as $S = [R_1]d$. Then, the decryption occur as: $\text{DEC}(R_1, R_2) = -S \oplus R_2 \equiv -[Qr]P \oplus (m \oplus [Qr]) \equiv [m]P$, where $-S$ is the additive inverse of S, and $-[Qr]$ is the multiplicative inverse in the curve $\mathbb{E}$.

Homomorphic property The point addition of two ciphertexts generates the sum of the clear texts, as we can see on:

$$\begin{aligned} R_1 \oplus R_1' &= [r]P \oplus [r']P \\ &= [r+r']P \\ R_2 \oplus R_2' &= ([m_1]P \oplus [r]Q) \oplus ([m_2]P \oplus [r']Q) \\ &= [m_1+m_2]P \oplus [r+r']Q \\ \text{ENC}(m_1) \oplus \text{ENC}(m_2) &= ([r+r']P, [m_1+m_2]P \oplus [r+r']Q \\ &= \text{ENC}(m_1+m_2) \end{aligned} \tag{3.7}$$

3.1.4 Benaloh

The work from (BENALOH, 1994) improved the Goldwasser-Micali scheme, presented in Subsection 3.1.2. Rather than ciphering one bit at a time, it is possible to encrypt a block of messages now. Their scheme is based on the difficulty of the residuosity problem (x^r), which is a generalization of the quadratic residuosity (x^2), where it is hard to find an x such that $z \equiv x^r \pmod{n}$. It is an additive homomorphic scheme, being also homomorphic by addition and multiplication over scalars. As it is possible to perform operations in blocks of size r, its ring's space is defined now in $\mathbb{Z}_r$ and not over $\mathbb{Z}_2$.

Key generation Select two large prime numbers p and q; select r, which is the cipher block size, such that $r|(p-1)$ and $\frac{(p-1)}{r}$ are relatively prime, and r and $(q-1)$ are relatively prime. We compute $n = pq$, and consider the usual Euler's Totient ϕ function as $\phi(n) = (p-1)(q-1)$. Select $y \in \mathbb{Z}_n^* = x \in \mathbb{Z}_n \mid \gcd(x,n) = 1$ such that $y^{\phi(n)/r} \mod n \neq 1$, namely y must be chosen from a multiplicative subgroup of integers modulo n, which includes numbers smaller than r and relatively prime to r. The public key is (y,n) and the private key is (p,q).

Encryption Generate a random number $u \in \mathbb{Z}_n^*$. For a message $m \in \mathbb{Z}_r$, the encryption is given by: $\text{ENC}(m) = y^m \times u^r \pmod{n}$.

Decryption Consider c the received ciphertext. To decrypt the ciphertext, we need to run a loop, to find an $i \in \mathbb{Z}_r$ such that $(y^{-i} \times c)^{\phi/r} \equiv 1$. Once we find i to solve the equation, the message will be the i itself, namely $m = i$.

Homomorphic property The multiplication of the ciphertexts results in the addition of the clear texts, as we can check on the following, for messages m_1, m_2:

$$\begin{aligned} \text{ENC}(m_1) \times \text{ENC}(m_2) &= (y^{m_1} \times u_1^r \pmod{n}) \times (y^{m_2} \times u_2^r \pmod{n}) \\ &= y^{m_1+m_2} \times (u_1 \times u_2)^r \pmod{n} \\ &= \text{ENC}(m_1 + m_2 \pmod{n}) \end{aligned} \tag{3.8}$$

3.1.5 Paillier

This scheme is based on a new approach called *composite residuosity problem*, rather similar to the residuosity problem, where it is hard to find a $y \in \mathbb{Z}_{n^2}^*$ such that $z = y^n \mod n^2$. In this case, z is said to be nth residue modulo n^2, if such y exists. It is an additive homomorphic scheme, being also homomorphic by adding and multiplying scalars to the ciphertext.

Key generation Calculate $n = pq$, being p and q large prime numbers, and the Euler's Totient $\phi = (p-1)(q-1)$, such that $\gcd(n,\phi) = 1$, and consider $\lambda = \text{lcm}(p-1, q-1)$. Take a random integer generator $g \in \mathbb{Z}_{n^2}^*$; consider a function $L(u) = (u-1)/n$ for every u in the subgroup $\mathbb{Z}_{n^2}^*$; g must be chosen such that $\gcd(n, L(g^{\lambda} \mod n^2)) = 1$. The public key is (n, g) and the private key is (p, q).

Encryption Choose a random integer $r \in \mathbb{Z}_{n^2}^*$. The encryption is given by: $\text{ENC}(m) = g^m \times r^n \pmod{n^2}$.

Decryption Consider c the received ciphertext. If $c < n^2$, the decryption can be done as: $\text{DEC}(c) = \frac{L(c^{\lambda} \pmod{n^2})}{L(g^{\lambda} \pmod{n^2})} \mod n$.

Homomorphic property The multiplication of two ciphertexts results in the addition of the clear texts. The homomorphism for messages m_1, m_2 can be done as it follows:

$$\begin{aligned} \text{ENC}(m_1) \times \text{ENC}(m_2) &= (g^{m_1} \times r_1^n \pmod{n^2}) \times (g^{m_2} \times r_2^n \pmod{n^2}) \\ &= g^{m_1+m_2} \times (r_1 \times r_2)^n \pmod{n^2} \\ &= \text{ENC}(m_1 + m_2) \end{aligned} \tag{3.9}$$

3.1.6 Naccache-Stern

As a generalization of Benaloh's cryptosystem, Naccache-Stern (NACCACHE; STERN, 1998) present two version schemes: one probabilistic and one deterministic. We will only show the probabilistic version, where they show improvements towards other previous homomorphic schemes. It is an additive homomorphic scheme, where it is also possible to perform addition or multiplication over an arbitrary scalar integer.

Key generation Select k numbers (being k an even number); then we select a family of k small odd distinct primes p_i. We set $u = \prod_{i=1}^{k/2} p_i$ and $v = \prod_{(k/2)+1}^{k} p_i$, and $\sigma = uv$. We also need two large prime numbers a and b, and calculate $p = 2 \times a \times u + 1$ and $q = 2 \times b \times v + 1$ are primes as well, and let $n = pq$. It is also necessary to find a random generator $g \in \mathbb{Z}_n^*$ such that the order of g is $\phi(n)/4$. The public key is (n, g, σ) and the private key is (p, q).

Encryption Generate a random number $x \in \mathbb{Z}_n$. The message $m \in \mathbb{Z}_\sigma$ is encrypted as: $\text{ENC}(m) = g^m \times x^{\sigma} \pmod{n}$.

Decryption Let $c = \text{ENC}(m)$. Consider the following list of congruence $m \equiv m_i \pmod{p_i}$ for $i = 1, \ldots, k$. Calculate for each ciphered element c_i the following: $c_i \equiv c^{\phi(n)/p_i} \pmod{n}$. Developing the mathematics, we arrive to the corresponding $c_i = g^{\frac{m_i \times \phi(n)}{p_i}} \pmod{n}$. By doing it for each c_i, we get the equivalent m_i, composing the original message.

Homomorphic property By multiplying the ciphertexts, we get the encryption of the sum of the original messages m_1, m_2:

$$\begin{aligned} \text{ENC}(m_1) \times \text{ENC}(m_2) &= (g^{m_1} \times x_1^{\sigma} \pmod{n}) \times (g^{m_2} \times x_2^{\sigma} \pmod{n}) \\ &= g^{m_1+m_2} \times (x_1 \times x_2)^{\sigma} \pmod{n} \\ &= \text{ENC}(m_1 + m_2) \end{aligned} \tag{3.10}$$

3.1.7 Okamoto-Uchiyama

The work of (OKAMOTO; UCHIYAMA, 1998) show a scheme, where the intractability is given by the hardness of finding p and q, given $n = p^2 \times q$. It is an additive homomorphic encryption scheme, where it is also possible to add or multiply to a scalar integer.

Key generation Choose large prime numbers p, q with a fixed bit size k and calculate $n = p^2 \times q$, where $\gcd(p, q-1) = 1$ and $\gcd(q, p-1) = 1$. Choose a random generator $g \in \mathbb{Z}_n^*$ with an order $g_p = g^{p-1} \mod p^2$. Let $h = g^n \mod n$. The public key is (n, g, h, k) and the private key is (p, q).

Encryption Consider the message $m \in [0, 2^{k-1}]$; select a random $r \in \mathbb{Z}/n\mathbb{Z}$, then the encryption is given as: $\text{ENC}(m) = g^m \times h^r \mod n$.

Decryption Let $c = \text{ENC}(m)$. Define a function $L(x) = \frac{x-1}{p}$ and an auxiliary $C_p = c^{p-1} \mod p^2$. The decryption can be performed as following: $\text{DEC}(c) = \frac{L(C_p)}{L(g_p)^{-1}} \mod p$, where $L(g_p)^{-1}$ is the modular inverse of the discrete logarithm function L.

Homomorphic property The function L and the encryption have the homomorphic property from multiplication to addition. This process is shown for messages m_1, m_2:

$$\begin{aligned} \text{ENC}(m_1) \times \text{ENC}(m_2) &= (g^{m_1} \times h^{r_1} \pmod{n}) \times (g^{m_2} \times h^{r_2} \pmod{n}) \\ &= g^{m_1+m_2} \times h^{r_1+r_2} \pmod{n} \\ &= \text{ENC}(m_1 + m_2) \end{aligned} \tag{3.11}$$

3.1.8 Damgard-Jurik

As a derivation from the Paillier cryptosystem, Damgard-Jurik (DAMGÅRD; JURIK, 2003) show a new approach that is length-flexible, i.e., it can easily handle messages of arbitrary length, not needing to change anything on the public or secret key. To show where it was engineered to use primarily, we will briefly explain the concept of zero-knowledge proof.

It is a protocol where a prover can prove to a verifier that some statement is true, and not provide anything else but the statement. Damgard-Jurik's scheme allows several users to use the same modulus, making it an efficient zero-knowledge proof for relations between ciphertexts of different public keys. It is an additive cryptosystem, and it is also possible to perform operations using scalars for both addition and multiplication. The original work shows two cryptosystems designed specifically for zero-knowledge proofs and mix-nets, so we will show here a generalization as shown in (KOÇ; ÖZDEMIR; ÖZGER, 2021).

Key generation Choose two large prime numbers p, q and compute $n = pq$ and $\lambda(n) = \text{lcm}(p-1, q-1)$. For a ring modulus s, choose a base $g \equiv (n+1)^j \times x \pmod{n^{s+1}}$ from $\mathbb{Z}^*_{n^{s+1}}$, where j is calculated as $\gcd(j,n) = 1$, and $x \in \mathbb{Z}^*_n$. Choose d such that $d \pmod{n} \in \mathbb{Z}^*_n$ and $d \equiv 0 \pmod{\lambda(n)}$. The public key is (n, g) and the private key is (d).

Encryption Convert the message m to integer values, where $m \in \mathbb{Z}_{n^s}$; take a random integer $u \in \mathbb{Z}^*_n$. We can encrypt a message as: $\text{ENC}(m) = g^m \times u^{n^s} \pmod{n^{s+1}}$.

Decryption Let $c = \text{ENC}(m)$. In order to decrypt c, we need to define a logarithmic function $L(z) = \frac{z-1}{n} \pmod{n^s}$. We can then, decrypt the message as: $\text{DEC}(c) = L(c^d \mod n^{s+1}) \times (L(g^d \mod n^{s+1}))^{-1} \pmod{n^s}$.

Homomorphic property The multiplication of the ciphertexts generates the addition of the plaintexts, as we can prove in the following, for messages m_1, m_2:

$$\begin{aligned} \text{ENC}(m_1) \times \text{ENC}(m_2) &= (g^{m_1} \times u_1^{n^s} \pmod{n^{s+1}}) \times (g^{m_2} \times u_2^{n^s} \pmod{n^{s+1}}) \\ &= g^{m_1+m_2} \times (u_1^{n^s} \times u_2^{n^s}) \pmod{n^{s+1}} \\ &= \text{ENC}(m_1 + m_2) \end{aligned} \tag{3.12}$$

3.1.9 Kawachi-Tanaka-Xagawa

In (KAWACHI; TANAKA; XAGAWA, 2007) it is shown the first PHE scheme being based on hard lattice problems. It is the first to encrypt multi-bit messages. It considers a *pseudohomomorphism* property, where we can apply operations using the algebraic mathematics behind lattices and Gaussian distributions. This cryptosystem is based on the Shortest Vector Problem (SVP) (see Subsection 2.3.1), but the homomorphic property here is also possible due to Gaussian distributions. In that, we encrypt 0's and 1's using distributions. Whereas it is based on the intractability of the SVP problem, we could define it to be quantum resilient, i.e., post-quantum safe.

A generic example of how the distributions of the encryption are given is represented in Figure 5 and Figure 6, where the terms are explained hereafter. Consider u a normal vector and the private key in this scheme. The difficulty relies, besides being over lattices, on the idea that we do not know $\| u \|^{-1}$, to know where to search for 0's and 1's, because it is based on different hyperplanes $H_0, H_1, \ldots$. Consider each hyperplane $H_i = \{x \in \mathbb{R}^n \mid \langle x, u \rangle = i\}$, where $x \in \mathbb{R}^n$ and

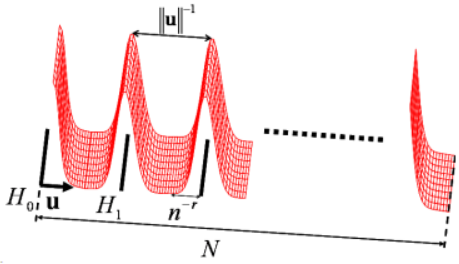


Figure 5 – Example ciphertexts of 0

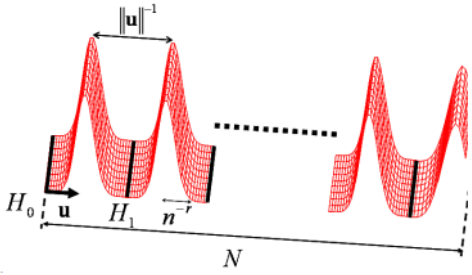


Figure 6 – Example ciphertexts of 1

Source: (KAWACHI; TANAKA; XAGAWA, 2007)

$i \in \mathbb{Z}$. Consider the following facts and why our secret key must contain u, from (KAWACHI; TANAKA; XAGAWA, 2007):

> We encrypt 0 into a vector distributed closely around hidden $(n-1)$-dimensional parallel hyperplanes $H_0, H_1, \ldots$ for a normal vector u of H_0 and encrypts 1 into a vector distributed closely around their intermediate parallel hyperplanes as $H_0 + \frac{u}{2\times\|u\|^2}, H_1 + \frac{u}{2\times\|u\|^2}, \ldots$

In order to understand the homomorphic property behind this scheme, we need to know about one property of Gaussian distributions: it is closed and homomorphic under addition operation. In this way, we represent a Gaussian distribution as $N(m, s^2)$ with mean m and standard deviation s. For distributions $X_1 = N(m_1, s_1^2)$ and $X_2 = N(m_2, s_2^2)$, we can calculate their sum as $X_1 + X_2 = N(m_1 + m_2, s_1^2 + s_2^2)$, i.e., it is homomorphic.

Preparation Let $N = n^n = 2^{n \log n}$. Define a n-dimensional hypercube $C = \{x \in \mathbb{R}^n \,|\, 0 \leq x_i < N, i = 1, \ldots, n\}$, an n-dimensional ball $B_r = \{x \in \mathbb{R}^n \,|\, \| x \| \leq n^{-r}/4\}$ for any constant $r \geq 7$. For $u \in \mathbb{R}^n$, define the hyperplanes $H_i = \{x \in \mathbb{R}^n \,|\, i = \langle x, u \rangle\}$. We must consider that $\| u \|$ should be in $[1/2, 1]$. It is necessary to set a parameter p, being the size of the clear text space. We consider the operation $\lfloor \rceil$ as the closest integer to the respective point, and ρ as the fundamental parallelepiped of a lattice (see Subsection 2.3.1).

Key generation Take a security parameter and size of n and a variance controlling r; choose the vector u uniformly at random from B_r. Let $M = n^3$. We need to sample M vectors $v_1, \ldots, v_M$, by repeating M times the procedure:

- Choose $a_i \in \{x \in C \,|\, \langle x, u \rangle \in \mathbb{Z}\}$ at random;
- Choose $b_1, \ldots, b_n$ from B_r uniformly at random;
- Make $v_i = a_i + \Sigma_{j=1}^{n} b_j$ as sample.
- Choose an index i_1 uniformly at random from $\{i \,|\, \langle a_i, u \rangle \not\equiv (0 \mod p)\}$

We take the minimum index i_0 satisfying that the width of $\rho(v_{i_0+1}, \ldots, v_{i_0+n})$ is at least $n^{-2} \times N$. The width of a parallelepiped $\rho(x_1, \ldots, x_n)$ is defined as:

$$min_{i=1,\ldots,n}(Dist(x_i, span(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)))$$

for a distance function $Dist(\cdot,\cdot)$ between a vector and an $(n-1)$-dimensional hyperplane. Let $w_j = v_{i_0+j}$ for every $j \in \{1,\ldots,n\}, V = (v_1,\ldots,v_M), W = (w_1,\ldots,w_n)$. We also need some auxiliary decryption information, which will be $k \equiv \langle a_{i_1}, u\rangle \pmod{p}$ The public key is (V,W,i_1) and the private key is (u,k).

Encryption We choose a uniformly random subset S of $\{0,1\}^M$, then we encrypt a message $m \in \{0,\ldots,p-1\}$ as follows: $\text{ENC}(m) = \frac{m}{p} \times v_{i_1} + \Sigma_{i\in S} v_i \pmod{\rho(W)}$.

Decryption Let $c \in \rho(W)$ be the received ciphertext. We need to use our auxiliary decryption information to retrieve that, in the following manner: $\text{DEC}(c) = \lfloor p \times \langle c,u\rangle \rceil \times k^{-1} \pmod{p}$, where k^{-1} is the inverse of $k \in \mathbb{Z}_p$.

Homomorphic property The homomorphism in this scheme is possible because ciphertexts are points in a Gaussian distribution. As we stated before, the sum of two points in this distribution is homomorphic. We will show the additive homomorphism, for messages m_1, m_2 and corresponding random subsets S_1, S_2:

$$\begin{aligned}
\text{ENC}(m_1) + \text{ENC}(m_2) &= \frac{m_1}{p} \times v_{i_1} + \Sigma_{i\in S_1} v_i \pmod{\rho(W)} + \frac{m_2}{p} \times v_{i_1} + \Sigma_{i\in S_2} v_i \pmod{\rho(W)} \\
&= \frac{m_1+m_2}{p} \times v_{i_1} + (\Sigma_{i\in S_1} + \Sigma_{i\in S_2}) \pmod{\rho(W)} \\
&= \text{ENC}(m_1+m_2)
\end{aligned} \tag{3.13}$$

4 SOMEWHAT HOMOMORPHIC ENCRYPTION

Somewhat Homomorphic Encryption allows one to perform two types of operations into the ciphertext, yet for a limited amount of time. It is an important variant for HE schemes; we can use it in many applications in the real world, and it is very practical. Although LFHE and FHE schemes can evaluate an arbitrary number of operations over ciphertext, it is not yet practical. LFHE schemes have a large size of public and private keys and take so long to evaluate simple operations. On the other hand, SHE schemes have a reasonable size for keys while allowing the evaluation of a limited number of operations efficiently.

The difference between SHE and LFHE is, mainly, focused on the amount of operations we can perform upon each one. While in SHE schemes we can only operate with a fixed amount of operations, LFHE schemes are specially made to perform an arbitrary number of operations, which it is put in the parameters of key generation of each scheme. If we want to perform, for instance, one thousand multiplications in LFHE, it must be possible, whereas it can become large and impractical in reality, but it is possible, while not using the bootstrap technique. In counter-proposal, SHE schemes are limited by default, and we cannot change the number of operations unless we use the bootstrap technique.

In the next sections, we will give an introduction to how we can use homomorphisms with operations that go beyond the addition and multiplication of ciphertexts. We also present the literature on SHE and the importance of Gentry's breakthrough work.

4.1 EVALUATING ANY OPERATION

An SHE is a scheme designed to perform one type of operation multiple times (mostly the addition operation), and another operation for a limited number of times (usually multiplication). One might wonder exactly how we can make any calculations apart from that, and the answer is with Boolean circuits. In the explanation about the PHE scheme of Goldwasser-Micali Subsection 3.1.2, we show it is homomorphic by XOR operations, and this also means it is homomorphic by addition operations. It is necessary to understand Boolean algebra, to be aware of the difficulty of implementing calculations over encrypted data.

We can denote the sum of two variables A and B in Boolean algebra as A *OR* B, or it could be represented as $A+B$. The multiplication of the same two variables would be as A *AND* B, or $A \cdot B$. Boolean algebra is based on truth tables, where for each variable we perform all the calculations for a specific operation. The truth tables for the operations *AND* and *OR* can be shown in Table 2 and Table 3, respectively. Another important operation is *NOT*, where we flip each bit, so $NOT\ 1 = 0$ and $NOT\ 0 = 1$. To define this operation in a variable, like $NOT\ A$, we denote it as $\overline{A}$.

With that, we can make all other operators, like the famous *NAND* operator. We can represent any Boolean expression by using *only NAND* operations, so *NAND* is said to be functionally complete. So, the logical expression of a *NAND*, for variables A and B, can be given

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Table 2 – And truth table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Table 3 – Or truth table

as $A\ NAND\ B \equiv A \uparrow B \equiv \overline{A \cdot B}$. With that is possible to re-do all other Boolean functions, for instance, $NOTA$ can be represented in $NAND$ operators as $A \uparrow A$. But as we saw, it was necessary to only use NOT, OR, and AND operators. In order to accomplish full-adder functions, which allow us to sum multiple numbers taking the carry from one unit to another, it is necessary a different operator, the XOR. This operator is represented as $\oplus$ in Boolean algebra, and it can be performed using the following formula: $A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$.

In this way, we can make all operations, as our first computers start performing, from multiple additions we perform multiplications, from multiple subtractions we perform divisions etc. It is very hard to think of a way to translate some complex functions to operators only using additions and multiplications, but it is possible. With that in mind, we must know that it is necessary to actually translate complex function equations to Boolean arithmetic, so we can actually perform that using HE schemes. There is some research on this area, as in (CHATTERJEE; SENGUPTA, 2015), where they show examples of transforming some usual operations like conditionals, divisions, comparisons, and less/greater compatible to HE schemes operations.

It is important to be aware of the necessity of transforming regular operations using Boolean algebra to use in HE schemes. This increases the complexity involved in the process of using HE for arbitrary calculations, as well as understanding the usage of some algorithms we will introduce later, as they are based on some specific circuits. It is also important to notice that the depth specified by these algorithms is related to the amount of circuit calculations it is going to be performed. It is still an open research field to know exactly how many operations will need to be performed upon some equation, whereas there are some techniques we can use to establish the number of multiplications necessary, for instance. In (PATERSON; STOCKMEYER, 1973), they show how we can make an upper bound of multiplications demanded. This is crucial to estimate, because as we said before, in probabilistic cryptographic homomorphic schemes, the noise grows after each operation over ciphertext, and with multiplication, this noise grows even more (see Subsection 2.2.1). With that in mind, we can have a good approximation of the number of operations that a SHE scheme can handle.

4.2 ENCRYPTION SCHEMES

The schemes related to SHE could be divided into two stages: the pre-Gentry era and the post-Gentry era. In Subsection 4.2.1, we will present the former, where it is possible to have

Table 4 – Summary of pre-Gentry schemes

Scheme	Underlying problem	Op1	Qnt1	Op2	Qnt2	Ciphertext	Space
BGN	Subgroup Decision	$+$	*	$\times$	1	Constant	$\mathbb{Z}_n$
SYY	Quadratic Residuosity	$\times$ (AND)	*	$+$ (OR/NOT)	1	Grows expo.	$\mathbb{Z}_n$
IP	(Protocol algorithm)	Branching programs				Grows expo.	$\mathbb{Z}_n$

$*$ in operations means an unlimited number of times; Grows expo is the abbreviation for "Grows exponentially"; *Op* stands for operation, and the index associated is to represent each operation over the ciphertext, as well as its corresponding amount (*qnt*).

two different operations for a limited amount of time. In the former, the schemes were thought to be FHE, using Gentry's bootstrap technique, being different in their creation and computations over ciphertext. Another point worth observing is the post-quantum assumption, where the majority of the post-Gentry era is based on Lattices, LWE, or RLWE, which are proven to be post-quantum secure. We will explain the main properties of each scheme, the difficulty it is bounded by, the depth of calculation for each circuit evaluation it can be performed, and whether the ciphertext grows for each operation or not.

4.2.1 Pre-Gentry era

We will present the main characteristics and procedures for each SHE of the pre-Gentry era, while a summary of the main properties is given in Table 4.

4.2.1.1 Boneh-Goh-Nissim

The BGN scheme (BONEH; GOH; NISSIM, 2005) was a breakthrough regarding operating with ciphertexts. It was the first that introduced, beyond multiple additions with ciphertext, one multiplication. Not only that, but it keeps the ciphertext size constant. Regarding the multiplication, it does not matter whether it is performed before or after the additions since it is based on bilinear groups. BGN is also able to perform 2-DNF formulas on ciphertext; DNF stands for Disjunctive Normal Form, i.e., a sum of products, or in-circuit evaluation notation, *OR* of *ANDs*. The algorithm intractability relies on the Subgroup Decision Problem, where it is hard to decide if an element is a member of a subgroup G_p of group G, with order $n = pq$, being p, q large prime numbers. With this scheme, it is also possible to multiply and add by arbitrary scalars.

Bilinear groups are used to make the one multiplication that can be performed over a ciphertext. We will give the principles of a bilinear group, as it is proposed in the original work (BONEH; GOH; NISSIM, 2005). Let $\mathbb{G}$ and $\mathbb{G}_1$ be two cyclic groups closed by multiplication of finite order n; g is a generator for $\mathbb{G}$. It is possible to calculate a bilinear map e, such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$, in other words, for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, we have $e(u^a, v^b) = e(u, v)^{ab}$. It is required that $e(g, g)$ is a generator for $\mathbb{G}_1$. In this case, $\mathbb{G}$ is a bilinear group.

Preparation We define the function $\mathscr{G}(\tau)$ to generate two random large prime numbers p,q, two bilinear groups $\mathbb{G},\mathbb{G}_1$ and a bilinear map e as a tuple, where τ is the number of bits the prime numbers must have. We then generate a bilinear group with the parameters as explained before (refer to (BONEH; GOH; NISSIM, 2005; KOÇ; ÖZDEMIR; ÖZGER, 2021) for details on the algorithm).

Key generation For a security parameter τ, we get the tuple $(p,q,\mathbb{G},\mathbb{G}_1,e)$ from $\mathscr{G}(\tau)$. Let $n = pq$. Select two random generators g,u for $\mathbb{G}$ and set $h = u^q$, such that h is a random generator of the subgroup $\mathbb{G}$ of order p. The public key is $(n,\mathbb{G},\mathbb{G}_1,e,g,h)$ and the private key is (q).

Encryption The message m must be an integer such that $m < q$. Let r be a random generator for $\mathbb{Z}_n = \{0,1,\ldots,n-1\}$ and calculate $\text{ENC}(m) = g^m \times h^r \in \mathbb{G}$.

Decryption Let $c = \text{ENC}(m)$. We can retrieve the original message with the formulas: $c^p = (g^m \times h^r)^p = (g^p)^m$.

$$\begin{aligned} c^p &= (g^m \times h^r)^p \\ &= (g^m)^p \times (h^r)^p \\ &= (g^m)^p \times ((u^q)^r)^p \\ &= (g^p)^m \times (u^{pq})^r \\ &= (g^p)^m \in \mathbb{G} \end{aligned} \tag{4.1}$$

Then we just need to calculate the discrete logarithm $\log_{g^p} c^p$.

Homomorphic property By multiplying the ciphertexts, we have the sum of the clear texts. For messages m_1,m_2, we can do the following:

$$\begin{aligned} \text{ENC}(m_1) \times \text{ENC}(m_2) &= (g^{m_1} \times h^{r_1}) \times (g^{m_2} \times h^{r_2}) \\ &= g^{m_1+m_2} \times h^{r_1+r_2} \\ &= \text{ENC}(m_1+m_2) \end{aligned} \tag{4.2}$$

In order to perform the one multiplication, it is necessary to change the base and the range of the encryption function once. Set $g_1 = e(g,g)$, $h_1 = e(g,h)$, being g_1 of order n and h_1 of order p. We have $h = g^{\alpha \times q}$ for some $\alpha \in \mathbb{Z}$. To multiply the ciphertexts c_1,c_2 for messages m_1,m_2, we need to pick a random $r \in \mathbb{Z}_n$ and set $c = e(c_1,c_2) \times h_1^r \in \mathbb{G}_1$. We can show the homomorphism in the following way:

$$\begin{aligned} c &= e(c_1,c_2) \times h_1^r \\ &= e((g^{m_1} \times h^{r_1}),(g^{m_2} \times h^{r_2})) \times h_1^r \\ &= g_1^{m_1 m_2} \times h_1^{m_1 r_2 + m_2 r_1 + \alpha \times q \times r_1 r_2 + r} \\ &= g_1^{m_1 m_2} \times h_1^{r'} \\ &= \text{ENC}(m_1 \times m_2) \end{aligned} \tag{4.3}$$

4.2.1.2 Sander-Young-Yung

SYY brings the idea of Goldwasser-Micali into a SHE scheme (SANDER; YOUNG; YUNG, 1999), so it is also based upon the Quadratic Residuosity Problem. It only allows encryption of binary numbers and it is multiplicative homomorphic (through *AND* operation), where it is also possible to perform one *OR* or one *NOT* operation. One downside of this scheme involves the ciphertext that grows by a constant multiplier at each OR/NOT operation performed over it. For simplicity of explanation, we will base our demonstration on the work of (KOÇ; ÖZDEMIR; ÖZGER, 2021). This scheme is an encoding representation of messages, that utilizes the Goldwasser-Micali encryption and decryption.

Key generation Let $n = pq$, for two large prime numbers p and q; choose a positive nonzero integer l and a quadratic non-residue $x \in \mathbb{Z}_n^*$, with $\frac{x}{n} = 1$, as demonstrated in the Goldwasser-Micali algorithm. The public key is (n, x, l) and the private key is (p, q).

Encoding The messages are bits, so the space is constrained to $\mathbb{Z}_2$ and, before the encoding begins, we need to make a map $\mathbb{E} \,|\, \mathbb{Z}_2 \to \mathbb{Z}_2^l$. With that, we will encode 1 with the zero vector in $\mathbb{Z}_2^l$ and encode 0 with a random nonzero vector in $\mathbb{Z}_2^l$. This is just a way of encoding the messages, we can decode it by applying a map "of return", like $\mathbb{D} \,|\, \mathbb{Z}_2^l \to \mathbb{Z}_2$, which decodes any nonzero vector to 0 and decodes the zero vector to 1.

Encryption After we encode the $m \in \mathbb{Z}_2$ to a vector $(v_1, \ldots, v_l) \in \mathbb{Z}_2^l$, we can encrypt it normally by using the standard Goldwasser-Micali encryption: $\text{ENC}(m) = \text{ENC}(\mathbb{E}(m)) = \text{ENC}(v_1, \ldots, v_l) = (\text{ENC}(v_1), \ldots, \text{ENC}(v_l))$.

Decryption We can decrypt the ciphertext using the same algorithm of decryption of Goldwasser-Micali (here represented as D), so we will only represent the vector encoded being decrypted: $D(\text{ENC}(v_1), \ldots, \text{ENC}(v_l)) = (v_1, \ldots, v_l)$. The vectors encoded can be converted back to bits using the map $\mathbb{D}$.

Homomorphic property Because of the encoding mechanism, we can check the *AND*-operation; if both encodings are the zero vector, they are both encodings of 1; if we multiply (*AND*) both, it will also result in the zero vector, that decodes to 1. The same can also be checked for the nonzero vectors (encodings of 0). Somewhat likely can be checked for *OR* and *NOT* operations. For more details on the proof, refer to (SANDER; YOUNG; YUNG, 1999).

4.2.1.3 Ishai-Paskin

The IP protocol (ISHAI; PASKIN, 2007) uses the encryption of the already explained Damgard-Jurik cryptosystem (see Subsection 3.1.8), therefore its underlying problem is also based on the Decisional Composite Residuosity problem. They proposed a special protocol,

that is able to evaluate branching programs based on representation models, such as circuits, formulas, OBDD (ordered binary decision diagrams, which consist of representing Boolean functions as directed acyclic graphs), finite automata, decision trees, and truth tables.

They propose the evaluation of a remote program P on an information x, such that $P(x)$ is only available to the owner of the information, and this person also cannot know any information about the remote program P. Actually, the final user would like to perform over a ciphertext of x, denoted as c. In that case, the server would perform the operations in a way that $P(c)$ solves to a solution c', which is also encrypted and only the user owner of the private key of the original c is able to decrypt it correctly. This approach would allow the evaluation of the medical history (x) of someone and P could be a complex program to decide whether the company would provide medical insurance or not. The same procedure could be used for predicting price algorithms of finance institutions (where we cannot know the algorithm and they should not know about our data); analogously, we could reference this to oblivious transfer algorithms to build protocols for secure multiparty computation. Their protocol could also be used to bind other protocols such as one-round keyword search and private information retrieval (PIR). These applications will be presented in more detail in Chapter 6.

Their solution was elegant in a way it is not possible to know any information, besides the size of the remote program P and the client's work on evaluating values should not be dependent on the size of P. With that, c' would only be size-dependent of the input x and the output of $P(c)$. As the algorithm of encryption is the same for Damgard-Jurik, we will not make the steps to produce the keys and ciphertexts, however, we will explain in a shallow way the protocol and what they did to perform on encrypted data.

They define a deterministic branching program P as a directed acyclic graph in which each node is represented as input variables and every nonterminal node has two outgoing edges, labeled 0 and 1, which should direct to another node. So, given an input $x \in \{0,1\}^n$ we would compute from an initial node to a terminal node, which would be the output of $P(x)$. The size of the program P is naturally the number of nodes in the graph and its length is defined as the longest path from the initial node to the terminal node. With that representation, we would be able to compute any problem in logarithmic space, or a P-problem, i.e., a family of polynomial-size branching programs. So, if any function can be converted to a deterministic finite automaton with s states, it can be computed by a branching program of length n and size $sn+1$. To explain their protocol, we will cite themselves:

> The ciphertext c is obtained by separately encrypting each bit of x using a homomorphic public key encryption scheme. (For efficiency reasons we rely on the Damgard-Jurik scheme; this scheme was previously used in the context of PIR by Lipmaa). To evaluate P on x we proceed in a bottom-up manner. Starting from the terminal nodes, in the i-th iteration we handle all nodes whose distance from the terminal nodes is i. For each such node, we compute a ciphertext containing an (iterated) encryption of its value. Using the homomorphic property, the encryption assigned

> to every node can be computed from the encryptions assigned to its children (which were computed in previous iterations) and the encryption of the input bit labeling this node. The ciphertext c is the (iterated) encryption assigned to the initial node. The client can recover $P(x)$ by applying iterated decryptions to c.

It must be noted that this protocol relies on the intractability of the cryptosystem used on it. They mention the HE scheme can be exchanged by others with similar properties of homomorphism, so as they use Damgard-Jurik, it is not post-quantum safe. Nevertheless, this can be changed, if the cryptosystem used could be changed for a post-quantum one.

4.2.1.4 Polly cracker schemes

In a way of showing multiple constructions over homomorphic encryption schemes, and for the sake of completeness, we will briefly show Polly-Cracker systems. It is given Polly-Cracker the name of schemes whose encryption is based on elements of an ideal; the intractability is based on the difficulty of computing Grobner bases (ALBRECHT et al., 2011), which is a special set of ideals in a ring over a field, where it is possible to deduce the dimension and the number of zeros of a finite ring.

With that in mind, the work of (FELLOWS; KOBLITZ, 1994) was the first attempt to use this property in a public key cryptosystem. This turns out to be an SHE scheme, enabling additions and multiplications over the ciphertext. Also, in (BARKEE et al., 1994) they proved to be a bad idea to use this type of basis for encryption and decryption. Others tried to create better schemes, hoping to overcome such difficulty, as proposed in (LY, 2006; ALBRECHT et al., 2011); but they were all broken (HEROLD, 2012). It is important to keep in mind these schemes were all homomorphic and considered to be SHE, for an unlimited number of additions and a specific amount of multiplications.

4.2.2 Post-Gentry era

Now we focus on SHE created after 2009, i.e., after Gentry's bootstrap proposal (GENTRY, 2009). In his first attempt to create an FHE scheme, he first created a SHE and turned it into an FHE with that approach. We recall that in Subsection 2.2.2, we explained briefly how this process is performed, where Gentry used a third public key called evaluation key, to transform a SHE scheme into a FHE one. Others after Gentry made the same process and we detach these schemes from the pre-Gentry, because of the evaluation of ciphertexts. Before, it was very clear how many operations, specifically the number of multiplications, could be done into the ciphertext before it would become unmanageable. However, with these new schemes, it may not be so simple to calculate this number. Because of rearrangements on the public and private keys, compression of some parts during the encryption process, and the possibility of having a new special key to perform computations over ciphertext (evaluation key), it is very hard

Table 5 – Summary of post-Gentry schemes

Scheme	Underlying problem	Relinearization key	Noise
DGHV	AGCD		Grows exponentially
BV	LWE		Grows by a constant
BFV	RLWE	✓	Grows by a constant
Smart-Vercauteren	Ideal lattices		
GSW	LWE		Grows doubly exponentially

Relinearization key indicates if there is a special key to handle the multiplication of ciphertexts efficiently. The noise column indicates how the noise grows at each operation, being able to analyze which scheme is more efficient.

to calculate the number of circuits necessary, before noise grows too much. The evaluation key in these cases is often called *relinearization key* because it decreases the impact of multiplications. One example of such usage is demonstrated in Subsubsection 4.2.2.3.

We are going to present the next schemes in a simpler way, highlighting their new perspectives and what they improved from previous schemes. We will show the underlying problem each scheme is based on and their special properties to handle operations over ciphertext. It is worth mentioning that, despite these schemes being SHE, in literature, they are only used in its "FHE mode", which means, they only use them with the bootstrap technique from Gentry. Nevertheless, we consider those schemes in the SHE section, because we can use them in their "raw version", i.e., without the usage of the bootstrap technique. In consequence, we will present them here, and generalize how many operations we can make over encrypted data with these schemes, by giving an expression based on some parameters of the scheme. A summary table of these schemes is provided in Table 5. All the schemes presented here are bootstrappable, i.e., they can become FHE with or without the usage of an evaluation key, which is peculiar to each scheme. Here we consider a relinearization key, which is used to reduce the noise growth in multiplications, or an evaluation key to become FHE.

4.2.2.1 DGHV

Here we recall the scheme already introduced in Subsection 2.2.1, where we explained the noise problem from probabilistic encryption schemes and how they are related to homomorphic encryption. There, we already explained how (DIJK et al., 2010) proposed a simple, secure, and fast SHE scheme to perform simple operations over ciphertext, in order to test the bootstrap approach after (see Subsection 2.2.2). They started creating a simple symmetric key and expanded to an asymmetric key scheme, based upon the hardness of the AGCD problem (see Subsection 2.3.2), which is considered to be post-quantum secure. Here, we do not need a relinearization key to perform multiplications.

They also give an approximation of how many Boolean circuits their SHE scheme is able to perform before it gets unmanageable, considering parameters of security for making the keys and the noise. Let η be the bit-length of the secret key, ρ be the bit-length of the noise,

and γ be the bit-length of the integers in the public key. Consider a secondary noise parameter $\rho' = \rho + \omega log\lambda$, where λ is a security factor and ω is the convention rough estimate of the order of the growth.

The noise exceeds the capability of decryption, considering a polynomial form of integer operations of degree/depth d from a function f and $|f|$ be the l_1 norm of the coefficient vector of f, when

$$d \leq \frac{\eta - 4 - \log|f|}{\rho' + 2}$$

As mentioned earlier, it is a little difficult to calculate the amount of operations, but we can check it is a small value. That's why it is only considered applications with slow-depth multiplications and degree of circuits for SHE schemes, as it is rather hard to stipulate the exact amount.

4.2.2.2 BV

This was one of the first schemes to be practical, considering the bootstrap approach for FHE schemes. The work of (BRAKERSKI; VAIKUNTANATHAN, 2011) is based on the LWE problem in lattices, which was already explained in Subsection 2.3.3. This problem can be reducible to the worst-case hardness of SVP, so it is considered to be quantum-safe. The construction is very simple and yet very powerful. They were able to achieve key-dependent message security (KDM), which is secure while encrypting polynomial functions of its own secret key. Because of this property, they managed to make a scheme whose public key size does not rely on the depth of the evaluation circuit. As in previous works, they made a SHE scheme first, and then, with the squashing and bootstrapping technique, they were able to transform it into an FHE.

We demonstrate the steps to make the symmetric version of this scheme, for the sake of simplicity and full demonstration of the process.

Key generation Considering a polynomial ring $R_q = \mathbb{Z}_q/\langle x^n + 1\rangle$, where n is a power of 2, i.e, all integer polynomials of degree up to $(n-1)$ and coefficients in $\mathbb{Z}_q$, so addition and multiplication are performed modulo $(x^n + 1, q)$. We also define a random Gaussian distribution, denoted by χ. We sample s as our secret key from our distribution χ, denoted as $s \xleftarrow{\$} \chi$.

Encryption Let $a \xleftarrow{\$} R_q$ and $e \xleftarrow{\$} \chi$. The encryption of a message m, which must be in the ring of polynomials $R_2 = \mathbb{Z}_2[x]/\langle x^n + 1\rangle$, is given by: $\text{ENC}(m) = (c_0, c_1)$, where $c_0 = -a$ and $c_1 = as + 2e + m$.

Decryption For given ciphertext (c_0, c_1), $\text{DEC}(c_0, c_1) = c_0 + c_1 \times s \pmod 2$.

Homomorphic property Consider messages m_1 and m_2. Addition can be presented normally. Multiplication, on the other hand, will need some tweaks, which are going to be explained

hereafter.

$$\begin{aligned}
\text{ENC}(m_1)+\text{ENC}(m_2) &= (-a_1, a_1\times s+2e_1+m_1)+(-a_2, a_2\times s+2e_2+m_2)\\
&= ((-a_1-a_2),((a_1+a_2)\times s+2\times(e_1+e_2)+(m_1+m_2)))\\
&= \text{ENC}(m_1+m_2)\\
\text{ENC}(m_1)\times\text{ENC}(m_2) &= (-a_1, a_1\times s+2e_1+m_1)\times(-a_2, a_2\times s+2e_2+m_2)\\
&= ((-a_1\times -a_2),(a_1\times a_2)\times s^2+2(e_1\times e_2)+(m_1\times m_2))\\
&\stackrel{*}{=} \text{ENC}(m_1)\times\text{ENC}(m_2)
\end{aligned} \tag{4.4}$$

Multiplication is not as simple as presented above, because it would take the secret key s to the power of 2; in that way, in order to have homomorphic multiplication, we need to add another element, to decrypt the ciphertext correctly. So, for ciphertexts c_1, c_2, we can make $c_1 \times c_2 = (c_{mult,0}, c_{mult,1}, c_{mult,2})$, being $c_{mult,2} = c_1 c_1'$, $c_{mult,1} = c_0 c_1' + c_0' c_1$ and $c_{mult,0} = c_0 c_0'$. We would end up with $c_{mult,0} + c_{mult,1} s + c_{mult,2} s^2$, which we can decrypt correctly using a ciphertext with the triple $c = (c_0, c_1, c_2)$ and decrypting like $\text{DEC}(c_1, c_2) = c_0 + c_1 s + c_2 s^2 \pmod 2$.

The correct decryption is dependent on a polynomial function f with maximum coefficient M and degree D is given by $M \times ((t \times r \times n)^{1.5})^D < \frac{q}{2}$, because of modular function over a ring. It is a very good improvement, as the noise only grows at a constant M, whereas in other schemes the noise grew exponentially. Although we showed the symmetric key scheme, there is also an equivalent to the conventional public key system, where they used an approach proposed from (LYUBASHEVSKY; PEIKERT; REGEV, 2010). Other schemes also transform their equivalent symmetric key to public key with this approach, as it is considered ideal and fast for LWE and RLWE schemes. It is worth mentioning that several improvements and techniques make this scheme very efficient, compact, and secure, being very much tested and even being proven to be standardized.

4.2.2.3 BFV

BFV cryptosystem (FAN; VERCAUTEREN, 2012) is one of the most famous schemes quoted to be standardized by (ALBRECHT et al., 2018), as it inherits from the BGV cryptosystem and improves some constructions. It translates BGV (BRAKERSKI; GENTRY; VAIKUNTANATHAN, 2014) (and (LYUBASHEVSKY; PEIKERT; REGEV, 2010)) to the RLWE problem, performing better relinearization (transforming a 3-ring element into a 2-ring one). For the FHE case, it was the first scheme to propose a modulus switch technique, which simplifies bootstrapping.

Modulus switch (MARCOLLA et al., 2022) converts a SHE into an FHE scheme, to reduce the space of the ciphertext to be more efficient. It transforms a ciphertext $c \mod q$ into another ciphertext $c' \mod p$, where p is sufficiently smaller than q. In this way, each element in

$\mathbb{Z}_q$ is converted into an element in $\mathbb{Z}_p$ by first multiplying it by p/q, and then taking the closest integer. This is done mainly because the noise in the ciphertext decreases.

In order to perform the relinearization, required for multiplications, it is necessary the special relinearization key. They also show two versions of how to compute that, and here we will present only version 2, as it performs better and requires a smaller number of multiplications and number of operations, which should be faster and more efficient.

Preparation We consider a distribution χ to be B-bounded limited by $[-B,B]$; also we consider some expression $[a+b]_q \equiv (a+b) \pmod q$. We denote by $a \leftarrow \chi$ that a is sampled from a distribution χ. We consider the nearest integer of a number a, with the notation $\lfloor a \rceil$; the infinity norm $||a||$ is defined as $\max_i |a_i|$; and the expansion factor of a ring R is defined as $\delta_R = \max\{\frac{||a\times b||}{||a||\times||b||} \mid a,b \in R\}$. We define R_q as the set of polynomials in R with coefficients in $\mathbb{Z}_q$.

Key generation Consider $s \leftarrow R_2$, $a \leftarrow R_q$, $e \leftarrow \chi$, for a B-bounded distribution χ. We consider some integer p and calculate $n = p \times q$. It is important to notice that q does not need to be a prime number, due to the LWE assumption, and not the factorization intractability. Sample $a' \leftarrow R_n$, $e' \leftarrow \chi'$. We also need a plaintext space, which we denote here as t, which also does not need to be prime. Now we have three keys: the public key is $([-(a\times s+e)]_q, a, t)$, the private key is (s), and the relinearization key is $([-(a'\times s+e')+p\times s^2]_n, a')$.

Encryption To encrypt a message $m \in R_t$, let $(p_0,p_1) = publicKey$, i.e., $p_0 = [-(a\times s+e)]_q$ and $p_1 = a$. Also sample $u \leftarrow R_2$ and two noise terms $e_1, e_2 \leftarrow \chi$. We calculate $\triangle = \lfloor q/t \rfloor$. The encryption is given as follows: $\text{ENC}(m) = c_t = ([p_0\times u + e_1 + \triangle \times m]_q, [p_1 \times u + e_2]_q)$.

Decryption Let (c_0,c_1) be the received encrypted text. We have

$$\text{DEC}(c_0,c_1) = \lfloor \frac{t\times[(c_0+c_1\times s) \mod q]}{q} \rceil \pmod t$$

Homomorphic property We show next the homomorphic addition of the ciphertexts $(ct_0,ct_1),(ct_2,ct_3)$ of two messages m_1 and m_2.

$$\begin{aligned}\text{ENC}(m_1)+\text{ENC}(m_2) &= ([p_0\times u+e_1+\triangle\times m_1]_q,[p_1\times u+e_2]_q)+([p_0\times u'+e_1'+\triangle\times m_2]_q,[p_1\times u'+e_2']_q)\\ &= ([p_0\times(u+u')+(e_1+e_1')+\triangle\times(m_1+m_2)]_q,[p_1\times(u+u')+(e_2+e_2')]_q)\\ &= \text{ENC}(m_1+m_2)\end{aligned} \tag{4.5}$$

In order to show the multiplication of the ciphertexts, as it grows to a 3-ring-elements ciphertext (like shown in BV Subsubsection 4.2.2.2), we will already split it into three

parts c_0, c_1, c_2, as follows:

$$\begin{aligned} c_0 &= \lfloor \frac{t \times (ct_0 \times ct_2)}{q} \rceil \mod q \\ c_1 &= \lfloor \frac{t \times (ct_0 \times ct_3 + ct_1 \times ct_2)}{q} \rceil \mod q \\ c_2 &= \lfloor \frac{t \times (ct_1 \times ct_3)}{q} \rceil \mod q \end{aligned} \tag{4.6}$$

Now, we need to use the relinearization key to remove the term s^2 from c_2 (which is given by $(ct_1 \times ct_3)$, which is in another ring element (that is the reason why we make s^2 in the evaluation key, to remove it in this step). Let (ek_0, ek_1) be the tuple of the elements of the evaluation key. Consider the following formula:

$$(c_{2,0}, c_{2,1}) = ([\lfloor \frac{c_2 \times ek_0}{p} \rceil]_q, [\lfloor \frac{c_2 \times ek_1}{p} \rceil]_q) \tag{4.7}$$

The final result of the multiplication of two ciphertexts can be now gathered and the output is $\text{ENC}(m_1) \times \text{ENC}(m_2) \equiv \text{ENC}(m_1 \times m_2) \equiv ([c_0 + c_{2,0}]_q, [c_1 + c_{2,1}]_q)$. Note that the noise from this relinearization process is smaller than the multiplication of the error terms.

The depth expected from this SHE scheme, for L levels of multiplications, is given as follows:

$$4 \times \delta_R^L \times (\delta_R + 1.25)^{L+1} \times t^{L-1} < \lfloor \frac{q}{B} \rfloor$$

It is a very good depth, as the limit of the distribution B can be greater and it will not affect the number of multiplications that we can perform over the ciphertext.

4.2.2.4 Smart-Vercauteren

In (SMART; VERCAUTEREN, 2010; SMART; VERCAUTEREN, 2014), Smart and Vercauteren give an ideal lattices-based scheme, where it is possible to use SIMD-like operations. SIMD stands for Single-Instruction-Multiple-Data, usually addressed by computer architecture, where with only one instruction, it is possible to perform parallel operations. With this approach, the recryption function, a method necessary in the bootstrap technique to transform a SHE into an FHE, becomes much faster and simpler. These improvements were also used in LWE-related schemes, as the authors said would be possible. The message m is constrained to $m \in \{0,1\}$.

To understand the completeness of the scheme, we will explain some properties of polynomials. A monic polynomial is a polynomial whose largest degree term is always one, i.e., in a polynomial of the form $ax^2 + bx + c$, a will be one. Also, we need to understand the concept of irreducible polynomials. An irreducible polynomial cannot be factored in smaller polynomials. Consider a non-constant polynomial $p(x)$ over a field $\mathbb{F}$. The polynomial $p(x)$ is irreducible if it does not exist $p_1(x), p_2(x), \ldots, p_n(x)$ where each constant of p_i has a degree smaller than $p(x)$. We also need to know about the function $resultant(a,b)$, which is the resultant of two

polynomials. The resultant of two polynomials is a polynomial expression of their coefficients that is equal to zero if and only if the polynomials have a common root (WOODY, 2016). It is a discriminant of the coefficients of the polynomials, i.e., the roots of the polynomial.

Another important topic to discuss, to understand the complete scheme, is the extended Euclidean algorithm *XGCD*. In *XGCD*, we use the Bézout identity, that states: for integers a and b, whose $gcd(a,b) = d$, there exist integers x and y such that $ax + by = d$. The $XGCD(a,b)$ algorithm computes both GCD (d) and Bézout's identity (x,y). Another property is the equivalence between $ax + by = \gcd(a,b)$ and $ax \equiv \gcd(a,b) \pmod{b}$. Here if the $\gcd(a,b) = 1$, the numbers are coprime (correlatively prime).

Setup Consider a polynomial of degree N, and two positive integers to define the ball's center: η and μ. Consider η and μ as security parameters. For a positive value r, we define one "ball" $\mathscr{B}$ centered at the origin as $\mathscr{B}_{\infty,N}(r) = \{\Sigma_{i=0}^{N-1} a_i x^i \mid -r \leq a_i \leq r\}$.

Key generation Choose a monic irreducible polynomial $F(x) \in \mathbb{Z}[x]$ of degree N. Repeat the following steps until p is prime:

- Select uniformly at random a polynomial $S(x)$ from $\mathscr{B}_{\infty,N}(\eta/2)$.
- Calculate the polynomial $G(x) = 1 + 2 \times S(x)$.
- Calculate the resultant $p = resultant(G(x), F(x))$.

Calculate $D(x) = \gcd(G(x), F(X))$ over $\mathbb{F}_p[x]$. Let $\alpha \in \mathbb{F}_p$ denote the unique root of $D(x)$. Apply the *XGCD* algorithm over $\mathbb{Q}[x]$ to obtain $Z(x) = \Sigma_{i=0}^{N-1} z_i x^i \in \mathbb{Z}[x] \,|\, Z(x) \times G(x) = p \pmod{F}(x)$. Let $B = z_0 \pmod{2p}$. The public key is (p, α) and the private key is (p, B).

Encryption Select uniformly at random a polynomial $R(x)$ from $\mathscr{B}_{\infty,N}(\mu/2)$. For a message m, we have the function $C(x) = m + 2 \times R(x)$. The encryption is given as $\text{ENC}(m) = C(\alpha) \pmod{p}$.

Decryption Let c be the received ciphertext. The decryption can be done as follows: $\text{DEC}(c) = (c - \lfloor c \times B/p \rceil) \pmod{2}$.

Homomorphic property Consider ciphertexts c_1, c_2 for messages m_1, m_2, respectively. Consider R_1, R_2 as the random polynomials and C_1, C_2 as the auxiliary function $C(x)$ in the encryption for c_1, c_2, respectively. The sum or multiplication of two ciphertexts must be performed $\pmod{p}$, as follows:

$$
\begin{aligned}
\text{ENC}(m_1) + \text{ENC}(m_2) &= (c_1 + c_2) \pmod{p} \\
&= (C_1(\alpha) + \pmod{p}) + (C_2(\alpha) + \pmod{p}) \\
&= (m_1 + 2 \times R_1(\alpha) \pmod{p}) + (m_2 + 2 \times R_2(\alpha) \pmod{p}) \pmod{p} \\
&= (C_1(\alpha) \pmod{p}) + (C_2(\alpha) \pmod{p}) \pmod{p} \\
&= ((m_1 + m_2) + (2 \times R_1(\alpha) \times R_2(\alpha))) \pmod{p} \\
&= \text{ENC}(m_1 + m_2)
\end{aligned}
\tag{4.8}
$$

$$
\begin{aligned}
\text{ENC}(m_1) \times \text{ENC}(m_2) &= c_1 \times c_2 \pmod{p} \\
&= (m_1 + 2 \times R_1(\alpha) \pmod{p}) \times (m_2 + 2 \times R_2(\alpha) \pmod{p}) \pmod{p} \\
&= (C_1 \pmod{p}) \times (C_2 \pmod{p}) \pmod{p} \\
&= (m_1 \times m_2 + 2R') \pmod{p} \\
&= \text{ENC}(m_1 \times m_2) \\
R' &= (m_1 \times R_2(\alpha)) + (m_2 \times R_1(\alpha)) + (2R_1(\alpha)R_2(\alpha))
\end{aligned}
\tag{4.9}
$$

4.2.2.5 GSW

The work of (GENTRY; SAHAI; WATERS, 2013) shows some novel properties concerning SHE and FHE schemes. The hard problem involving this scheme is the LWE problem in lattices. Different from others, rather than performing additions and multiplications over a ring, which can cause an expansion factor of the ring elements (which should be solved using relinearization techniques), GSW is based on matrix additions and multiplications. As in relinearization, it is necessary to address three ring elements. This is considered to have a complexity of $\Omega(n^3)$, while matrix operations can be roughly optimized using Strassen and Williams, arriving at a complexity of $O(n^{2.3727})$ (WILLIAMS, 2012). As it does not contain the relinearization step, it does not need a relinearization key. The replacement for this procedure is, what the authors called the *approximate eigenvector* method, where the secret key is an *approximate eigenvector* of the ciphertext, while the original value is the *eigenvalue*. In linear algebra, we have a similar concept, where an eigenvector of a linear transformation is a nonzero vector that changes by a constant factor, where the eigenvalue is the multiplying factor.

Besides that, this is an Identity-based encryption (IBE) scheme, i.e., given a public ID representation of the user, it is possible to encrypt data and perform homomorphic operations over that. They also mention it is possible to extend this scheme to Attribute-Based encryption (ABE), which is similar to IBE, but we can make attributes for the public ID representation; this can be used for access control for determined authorization policies, namely, it is only possible to decrypt some ciphertext if its encryption contains the attributes of the public key encrypted altogether. There could be some rearrangements in the scheme to support multi-attributes for ABE in homomorphic evaluations, for instance.

Setup Choose a modulus q of $\kappa = \kappa(\lambda, L)$ bits, where λ is a security parameter and L is the maximum multiplicative depth supported, with lattice dimension $n = n(\lambda, L)$ and error distribution $\chi = \chi(\lambda, L)$ (over $\mathbb{Z}$ and with maximum bound B), so that the LWE problem achieves at least 2^λ security. Choose parameter $m = m(\lambda, L) = O(n \log q)$. Let $\ell = \lfloor \log q \rfloor + 1$ and $N = (n+1) \times \ell$.

Functions As this scheme uses multiple ideas for different values in the process, we will present some useful functions.

BitDecomp($\vec{a}$) is the N-dimensional vector $(a_{1,0},\ldots,a_{1,\ell-1},\ldots,a_{n,0},\ldots,a_{n,\ell-1})$, where $a_{i,j}$ is the j-th bit in a_i's binary representation (LSB as Least Significant Bits); consider the inverse $BitDecomp^{-1}(\vec{a}) = (\Sigma_{j=1}^{\ell-1} 2^j \times a_{1,j},\ldots,\Sigma_{j=1}^{\ell-1} 2^j \times a_{n,j})$.

Flatten($\vec{a}$) $= BitDecomp(BitDecomp^{-1}(\vec{a}))$.

Key generation Sample $\vec{t} \leftarrow \mathbb{Z}_q^n$, $\vec{s} \leftarrow (1,-t_1,\ldots,-t_n) \in \mathbb{Z}_q^{n+1}$, $\vec{v} = \vec{s} \times \vec{s}$ (consider this as the *eigenvector*). Generate a matrix $B \leftarrow \mathbb{Z}_q^{m \times n}$ uniformly and a vector $\vec{e} \leftarrow \chi^m$. Set $\vec{b} = B \times \vec{t} + \vec{e}$. Set A to be the $(n+1)$-column matrix consisting of $\vec{b}$ followed by n columns of B (here, we can denote that $A \times \vec{s} = \vec{e}$). The public key is (A) and the private key is $(\vec{s})$.

Encryption To encrypt a message $m \in \mathbb{Z}_q$ (*eigenvalue*), sample a uniform matrix $R \in \{0,1\}^{N \times m}$. Then, $\text{ENC}(m) = Flatten(m \times I_n + BitDecomp(R \times A)) \in Z_q^{NxN}$, where I_n is the N-dimensional identity matrix.

Decryption Consider that $q = 2^{\ell-1}$ and the first $l-1$ coefficients of $\vec{v}$ are $(1,2,\ldots,2^{l-2})$. Let c be the ciphertext. So $c \times \vec{v} = m \times \vec{g} + smallError$, where $\vec{v} = (1,2,\ldots,2^{l-2})$. Then, for each least significant bit from the message m from $m \times 2^{l-2} + smallError$, take the last bits and re-make the original vector of the message m.

Homomorphic property We only need to show the homomorphism in the Flatten function. For ciphertexts c_1, c_2 of original messages m_1, m_2, consider the following statements:

$$\begin{aligned}
\text{ENC}(m_1) + \text{ENC}(m_2) &= c_1 + c_2 \\
c_1 &= Flatten(m_1 \times I_n + BitDecomp(R_1 \times A)) \\
c_2 &= Flatten(m_2 \times I_n + BitDecomp(R_2 \times A)) \\
bit_1 &= BitDecomp^{-1}(m_1 \times I_n + BitDecomp(R_1 \times A)) \\
bit_2 &= BitDecomp^{-1}(m_2 \times I_n + BitDecomp(R_2 \times A)) \\
bit_1 + bit_2 &= BitDecomp^{-1}(m_1 \times I_N + BitDecomp(R_1 \times A)) + (m_2 \times I_N + BitDecomp(R_2 \times A)) \\
&= BitDecomp^{-1}(m_1 \times I_N + m_2 \times I_N BitDecomp(R_1 \times A)) + BitDecomp(R_2 \times A) \\
c_1 + c_2 &= BitDecomp(bit_1 + bit_2) \\
c_1 + c_2 &= BitDecomp(BitDecomp^{-1}((m_1 + m_2) \times I_N + BitDecomp(R_1 \times R_2 \times A))) \\
&= \text{ENC}(m_1 + m_2)
\end{aligned} \tag{4.10}$$

$$\begin{aligned}
\text{ENC}(m_1) \times \text{ENC}(m_2) &= c_1 \times c_2 = Flatten(c_1 \times c_2) \\
&= c_1 \times c_2 \times \vec{v} \\
&= c_1 \times (m_2 \times \vec{v} + \vec{e_2}) + m_2 \times (m_1 \times \vec{v} + \vec{e_2}) + c_1 \times \vec{e_2} \\
&= m_1 \times m_2 \times \vec{v} + m_2 \times \vec{e_1} + c_1 \times \vec{e_2} \\
&= \text{ENC}((m_1 \times m_2) \times \vec{v})
\end{aligned} \tag{4.11}$$

In order to know how many operations we can perform in the ciphertext, we consider the multiplicative depth L of the circuit to be evaluated and a maximum bound B from the random

distribution used. For that, the error is considered to grow B^{2^L}, so q must be smaller than this. The authors also check the possibility of evaluating using the degree of functions to be evaluated (and not the depth itself). So, if we consider a multivariate polynomial $P(x_1,\ldots,x_t)$ of degree d, the last ciphertext possible is bounded by $|P|(N+1)^{d-1}B^d$.

5 OTHER TYPES OF HOMOMORPHIC ENCRYPTION

Previously, we discussed the basic types of homomorphic encryption, where it all began, and how the schemes and implementations evolved throughout the years. First, we were only able to perform one type of operation on the ciphertext, using PHE, whereas in SHE some arrangements changed. In SHE, more complex applications were reached with some more sophisticated mathematics behind reaching one type operation as many times as necessary and another type for a limited pre-determined number, or even, a protocol that makes it possible to evaluate branching programs (see Subsubsection 4.2.1.3).

In Chapter 6, we will present the main applications for each variant. There are many applications achievable with HE nowadays using only PHE and SHE, with implementations that are already being used today but may not be using their full power, and protecting people's data 100% of the time. Although these variants are very important and were the beginning of HE, there are more elaborated constructions, as is the case for LFHE and FHE. In this chapter, we will present the main characteristics of general LFHE and FHE, however not showing specific schemes' constructions.

5.1 LEVELED FULLY HOMOMORPHIC ENCRYPTION

Leveled Fully Homomorphic Encryption is a very interesting and important variant of HE because we can really apply it to real complex applications. Some very fascinating privacy-preserving applications, at low depth of evaluation, have already been implemented by multiple developers all over the world just using LFHE. We consider a scheme to be LFHE if we can determine the depth we can reach of evaluations, so the decryption is still correct. As we do not know exactly when and at which operation we will make noise greater than it should, some heuristic models are applied, for the purpose of guessing this probability.

Because of that, some differences are found over other schemes that were presented before, as the depth can be configured in the key generation process. The schemes of this variant are faster than FHE in terms of operation evaluation and, for those that support bootstrapping, are also considered to be faster and more efficient. It is important to mention that some schemes, in order to become fast enough, use a precision evaluation of computations, so it might contain an inaccuracy in the processing of operations over the ciphertext, which also is parameterized at some moment in the scheme. The arithmetic operations then are only done on the most significant bits (MSB), then the result may be rounded using some of LSB, to keep the bit size (*mantissa*). One example of such a scheme is CKKS, which will be briefly presented hereafter.

As this variant is mainly based on a limit of depth, applications squeeze the amount of operations necessary, to not grow the noise too much, and be able to still use some powerful and fast LFHE schemes. As stated previously on Subsection 2.2.2, the main search for new, more compact, and more efficient algorithms in HE now are based on, normally, two different approaches: 1) fast bootstrapping and 2) leveled approach. In the former, we try to perform the

bootstrap technique faster with better data structures and algebra tricks. Some examples of fast bootstrapping consider the already explained GSW (see Subsubsection 4.2.2.5), and two already not referred to before, the FHEW (DUCAS; MICCIANCIO, 2015) and TFHE (CHILLOTTI et al., 2020). Although FHEW and TFHE are specialized in fast bootstrapping, they are LFHE. The last approach tries to get better approximation error, so the ciphertext does not get unmanageable. Leveled Fully Homomorphic Encryption refers to the second approach.

A summary of some schemes and their properties can be seen in Table 6. It is worth mentioning that all the LFHE schemes presented here are bootstrappable, which means, all schemes can become FHE (with or without the usage of an evaluation key). A brief description of each one is given next:

- Melchor-Gaborit-Herranz (MELCHOR; GABORIT; HERRANZ, 2010) is done by constructing a theoretical object by chaining different encryption schemes and combining their homomorphic properties. In the work, it is shown how a combination of BGN (see Subsubsection 4.2.1.1) and Kawachi (see Subsection 3.1.9) was able to perform a pre-established amount of multiplications over the ciphertext. It is also possible to combine different cryptosystems since some properties are satisfied.

- The CKKS scheme (CHEON et al., 2017) uses number approximation arithmetic and also allows to perform operations over real numbers. In contrast, in previous schemes all operations and rings were only made upon integers. This scheme is, in particular, very important for LFHE, as it is very fast to make calculations upon and very powerful, showing on their paper that it is computationally efficient to perform evaluations of multiplicative inverse, exponential function, logistic regression, and discrete Fourier transform, besides being able to evaluate a big amount of multiplications.

- FHEW (DUCAS; MICCIANCIO, 2015) was the first scheme to propose a fast-way for bootstrapping technique, claiming to be able to bootstrap in half a second. The authors first construct an LFHE scheme and make improvements on bootstrap through this.

- TFHE (CHILLOTTI et al., 2020) is an improvement of FHEW, by making the LWE representation in a mathematical torus. Zama, which is a group of researchers that made TFHE, has some very interesting and important applications as open-source on their GitHub platform, which shall be described later on Section 6.3.

- Doroz-Sunar (DS) (DORÖZ; SUNAR, 2020) is an NTRU-based scheme, which is not vulnerable to multiple attacks on these types of schemes. There is no evaluation key to bootstrap, the noise grows linearly and the ciphertext is constant.

- Armknecht-Sadeghi (AS) (ARMKNECHT; SADEGHI, 2008) shows us a new way of making homomorphic schemes, by using coding theory hard problems, and it can compute over infinite fields (such as rational numbers). There is no evaluation key to perform

Table 6 – Summary of LFHE schemes

Scheme	Underlying problem	Evaluation key	Noise growth	Ciphertext
Melchor	SVP		Exponential	Grows
CKKS	RLWE	✓	Linear	Depends on depth
FHEW	LWE	✓	Constant	Constant
TFHE	LWE over Torus	✓	Constant	Constant
DS	NTRU		Linear	Constant
AS	Reed-Solomon codes		Limited to depth	Grows exponentially
DK	LWE with Multivariate polynomials		Linear	Constant

The column evaluation key marks if the scheme uses an evaluation key to reach FHE. Noise growth refers to how the noise grows at each homomorphic evaluation. Similar to the ciphertext, where we denote which schemes increase the size of the ciphertext at each homomorphic operation.

the bootstrap technique. The noise grows depending on the depth chosen, whereas the ciphertext grows exponentially at each multiplication;

- Dowerah-Krishnaswamy (DK) (DOWERAH; KRISHNASWAMY, 2019) uses the hardness of LWE, combined with retrieving multivariate polynomials, not requiring relinearization, nor an evaluation key. The ciphertext remains constant for each homomorphic operation.

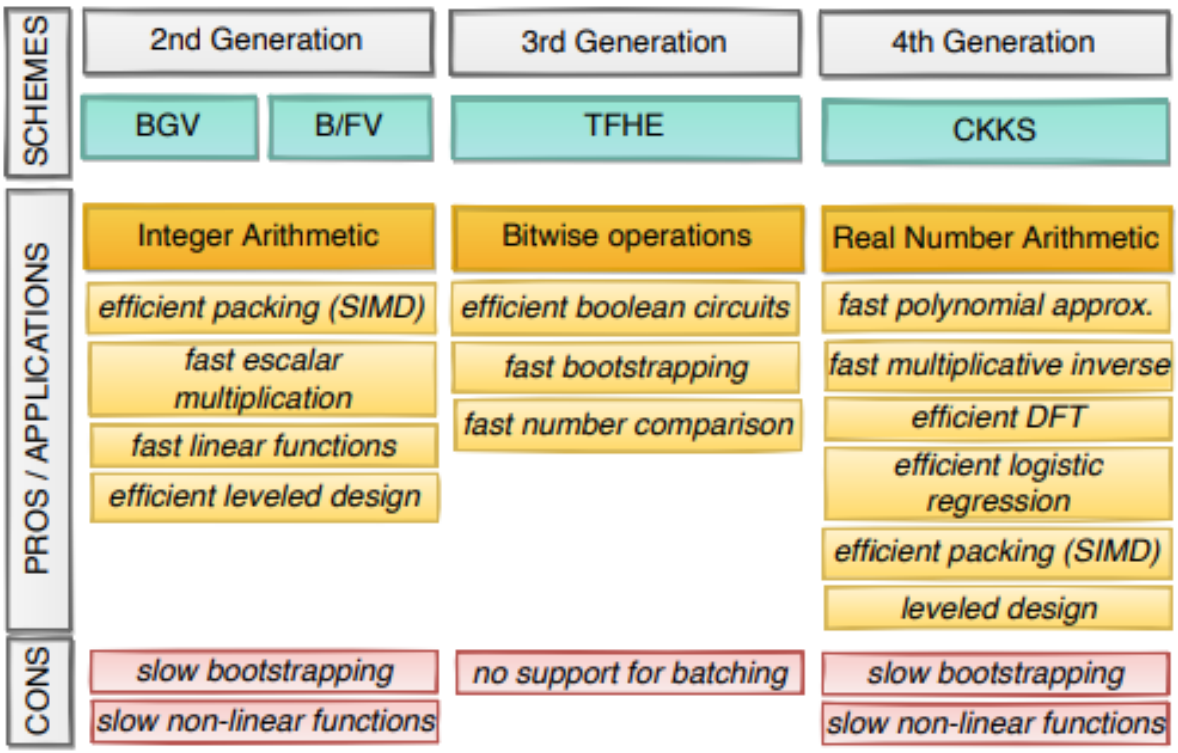
5.2 FULLY HOMOMORPHIC ENCRYPTION

FHE is considered to be the Holy Grail of calculations over encrypted data. With FHE, it is theoretically possible to perform any calculation over any encrypted data, while still preserving the ciphertext size. Gentry in 2009 (GENTRY, 2009) proposed the first FHE scheme, with his bootstrap technique. However, until today we do not have a practical FHE scheme, that is both compact and fast to perform calculations.

One way of following schemes' evolution in FHE schemes is by the so-called generation of the schemes. Since 2009, researchers have categorized each scheme, as some progress was reached. Figure 7 shows a brief summary of the main improvements throughout the years and generations. We recall that the presented post-Gentry (see Subsection 4.2.2) and LFHE (see Section 5.1) schemes are bootstrappable, i.e., it is possible to make them FHE. In Figure 7, we can check the second generation is mainly SHE schemes, while the third is focused on fast bootstrapping, and the last is focused on the leveled approach. A succinct explanation of each generation will be given next:

- The first generation contains only DGHV (see Subsubsection 4.2.2.1), being the first viable example after Gentry's bootstrap publication;
- The second generation comes with great improvements, containing BV (see Subsubsection 4.2.2.2), BGV (see (BRAKERSKI; GENTRY; VAIKUNTANATHAN, 2014)), BFV

Figure 7 – FHE generation of schemes and explanations



Source: (MARCOLLA et al., 2022)

(see Subsubsection 4.2.2.3), the multi-key NTRU-based scheme (LÓPEZ-ALT; TROMER; VAIKUNTANATHAN, 2012), GSW (see Subsubsection 4.2.2.5), and Smart-Vercauteren (see Subsubsection 4.2.2.4). They still follow Gentry's construction of first developing a SHE and transforming it into an FHE through bootstrap. Some optimizations in key switching and modulus switching were proposed, as well as the parallel instructions (SIMD);

- The third generation is based on the advances in making fast-bootstrapping procedures, as the ones in FHEW (DUCAS; MICCIANCIO, 2015) and TFHE (CHILLOTTI et al., 2020). The process of bootstrapping now can cost only half a second on desktop computers.

- The fourth generation represents today's state of the art, as we port FHE to real applications. It is marked by the arrival of the CKKS scheme (CHEON et al., 2017), which performs operations on rounded values, instead of exact numbers. This allows a better management of the noise and allows the usage of machine learning, where it handles data in approximation forms.

In (MARCOLLA et al., 2022), the authors describe libraries, compilers, and hardware accelerators that implement and make all the optimizations described in the most recent literature. Much progress has been made in key sizes (in the beginning was about 1GB, now we have something around 10MB), time on performing operations on ciphertext, and bootstrapping. Also,

large companies like Microsoft (RESEARCH REDMOND, 2023), IBM (FULLY...,), and Google (GUEVARA, 2023) are making big efforts to make it happen, in order to get more privacy for their users and be considered more reliable by the public.

6 APPLICATIONS

6.1 PARTIALLY HOMOMORPHIC ENCRYPTION

Applications for PHE schemes are limited, due to having only one type of operation that can be performed over the ciphertext. Nevertheless, some important privacy-preserving applications can make use of that. Some examples are online voting systems, which can calculate the number of votes for each participant in the election by homomorphically adding all the votes; blockchain ledgers, whose only operation needed to know the exact balance or the transaction history is adding the past transactions, which can be made homomorphically; in (SAVIĆ et al., 2018), they demonstrate that for some specific IoT areas, such as smart home sensors, it is possible to perform transactions between cloud-computer-sensors using PHE encryption and only call the sensor again after some server calculations and over scalars.

Applications mentioned in (BENALOH, 1994) consider verifiable secret sharing and verifiable secret ballot elections. The first can be considered as a way to divide a secret into shares, for different shareholders (where each one holds a secret key). The only way to reconstruct the complete secret is by calculating a specific amount from n shareholders. The authors show that for a determined polynomial, the only operation necessary to reconstruct the secret is by performing the addition of each part of it, in this case, each part of the secret, i.e. the share of each shareholder. It is verifiable, so one dishonest part cannot disrupt the reconstruction of the secret, as it can only be reconstructed if a sufficient amount of trusted shareholders give their own share. The verifiable secret-ballot elections are similar to the idea presented before, where a central entity creates an additive probabilistic PHE and makes the public key, public. In that way, each voter can compute, for a new random value, the encryption of yes (1) or no (0) and put it into the ballot. Then, the central entity, which possesses the private key only needs to sum all the votes and homomorphically decrypt it, revealing the result of the election.

In (NACCACHE; STERN, 1998; PFITZMANN; SCHUNTER, 1996), they propose to put a watermark on some value or product. Aiming the software industry or illegal image and music reproduction, it is possible to create a watermark, so that only the buyer knows the data with the fingerprint. Still, if the merchant finds this copy somewhere else, he can identify the buyer and prove that the determined buyer bought the copy and is, perhaps, illegally distributing it. The computations can be made homomorphically and it only needs to perform additions over the ciphertext.

In (CRAMER; DAMGÅRD; NIELSEN, 2001), it is proposed the usage of PHE like Paillier's for MPC. Multiparty computations rely on the multi-processing of data between different peers, while the peers only know pieces of information about the original data. Originally, it was proposed the use of Shamir's secret sharing (SHAMIR, 1979) to divide the information between n peers, in a way that can only be reconstructed together using a specific amount of the shares. However, in MPC we can send one piece to each party, encrypted; then each party performs a calculation; at the end, the encrypted result of all peers is added together homomor-

phically and then decrypted. Only the owner of the data possesses the secret key to decrypt it and get the calculation together of the peers.

One other application of PHE is keyword search. The work of (AMORIM; COSTA, 2023) shows that there are several searchable schemes that use raw PHE schemes and they are very efficient. They gathered multiple schemes that can do such computation efficiently and classified what strategies each uses. They also showed that multiple strategies consider PHE to reach keyword searches. However, they can execute only some raw operations such as single-keyword search in sequential scanning for single users. In (SILVA, 2016), it is implemented a file search application, using already established homomorphic libraries.

Yet another example is the private information retrieval protocol (PIR), which relies on the problem that, given a public database and a user wanting to query some information on it, it is difficult to hide that query from the database's operator. The protocol defines a way for the user to encrypt the query, and only the user has the public key to decrypt it. There are multiple implementations of such protocol, where it is necessary to iterate the whole database to perform a hidden query. Some of these arrangements can be constructed only by using an additive PHE with support for scalar multiplications.

6.2 SOMEWHAT HOMOMORPHIC ENCRYPTION

We will give some ideas of projects, protocols, equations, and general applications we could be using SHE. We need to keep in mind that with SHE, we cannot use so many multiplications, because all schemes only support a small number of them. So we need to think of ways to minimize the amount of this operation or substitute them with an equivalent operation; for instance, if the user wants to perform a calculation that performs a low-depth of multiplications, we can substitute them for multiple sums of the number (it is necessary the interaction with the user); or, alternatively, if the user needs to multiply by a constant, that could not affect the retrieval of whole information, we could use a scheme that allows to do it using multiplication by a scalar.

In (NAEHRIG; LAUTER; VAIKUNTANATHAN, 2011), authors consider some systems that require many additions and only a small number of multiplication over ciphertexts. They consider applications in some very important areas throughout the world, like medical, financial, and advertisements. They consider both functions that should operate over sensitive information and functions that are proprietary. They also mention some very important calculations that are used almost anywhere and could be performed only with SHE: average (which requires no multiplication), standard deviation (only one multiplication required), and logistical regression (only a small number of multiplications, depending on the precision required). If we have logistical regression, we can evaluate a wide variety of applications that require Big Data and artificial intelligence. One problem that is also mentioned is the constraint of not knowing how to divide real numbers or take square roots with these schemes. All of the aforementioned applications then should return the encryption of the numerator and the encryption of

the denominator, then the user decrypts it and makes the calculation herself.

Although not very practical, it is important to be aware of the existence of such possibilities, so that in the future, we might resolve it somehow. In (ILIASHENKO; NEGRE; ZUCCA, 2021), they demonstrate some interesting applications using only SHE in some cases, such as the parity function (checking if a number is odd or even), if a number is the power of another, taking the modulo of a number, taking the Hamming weight (accounts the number of symbols that are different from zero) and, also, the Hamming distance (number of positions different from two strings), performing the conditional less-than function, and indicate how to perform the operation $c \pmod 2$, being c a ciphertext. These functions can be performed using much fewer multiplications because some techniques are applied involving finite fields theory.

Some more complex applications can be formed, with many more operations and function evaluations. We cannot forget that SHE schemes are a super-set of applications within the context of PHE. So all the applications explained earlier can be combined with these, as a way of making more elaborated utilization. Also, we cannot forget the branching programs introduced by Ishai-Paskin (ISHAI; PASKIN, 2007) (see Subsubsection 4.2.1.3), which enables us to a variety of domains.

As stated previously in PHE applications (see Section 6.1), we can make PIR protocols using only PHE. There are multiple implementations using homomorphic encryption over whole databases and performing queries over ciphertexts. Still, they are very inefficient and also need to handle the whole database, with the additional overhead of HE for itself. In (PARK; TIBOUCHI, 2020) is proposed a small and contained way for PIR protocol, where it is only necessary to iterate $O(\log n)$ items from the database, and it is very compact. They use compressing techniques of ciphertexts in the database and over the user's query using a SHE.

In the work of Ishai-Paskin (ISHAI; PASKIN, 2007) (see Subsubsection 4.2.1.3), it is proposed a protocol that enhances the ability of keyword search (as it is already achievable with PHE). With that, it is possible to completely hide from the client the original size of the database, whereas it is also possible to hide from the database operator anything related to the client's query. They can perform these tasks in one round, i.e., it is not necessary to use a three-way handshake as in the majority of other schemes.

6.3 LEVELED FULLY HOMOMORPHIC ENCRYPTION

There are many different applications for LFHE since we can evaluate a low depth of multiplications, but larger than the ones available at SHE. It is worth mentioning that LFHE embraces both PHE and SHE applications, and yet is able to perform others. One difficulty while thinking about applications for LFHE, is the case where we need to calculate the exact parameters for a determined scheme to have enough operations, without the noise to get unmanageable. Also, we need to keep in mind that, after the noise gets too large, we would need to bootstrap (which would be an FHE). So, applications that require a small (but higher than SHE, normally) amount of multiplications can be used with this variant.

For the next example, we need to know what group testing is. This is a procedure to identify different “objects”, using combinatorial mathematics, in a large number of objects by grouping them into small groups and testing them. An example involving CKKS and group testing can be found in (IBARRONDO et al., 2023), where they use CKKS for privacy-preserving face identification. As this type of problem deals with people’s security, it is important to keep all the information in the process hidden from potential attackers. The paper describes how it is necessary to compare multiple data (the user’s input and the one registered), in order to ensure face identification. As these operations are expensive to perform using only additions and multiplications, they surpassed this problem by using group testing, and decreasing the amount of multiplications necessary.

Sometimes we need to rely on other problem categories to decrease the amount of operations necessary to handle encrypted data, using the multiple variants of HE. In (LOU; JIANG, 2019), it is given an example of how to achieve a neural network on encrypted data using leveled TFHE. The authors used a special type of operation called logarithmic quantizations, to replace expensive multiplications, being able only to use native shift operation, already implemented into the TFHE library. They also implemented a rectified linear (ReLU) activation function, and max poolings using raw operations of the scheme. As we mentioned, this is also another problem for HE schemes, to reimplement the already known algorithms, to have only raw and supported operations.

6.4 FULLY HOMOMORPHIC ENCRYPTION

The range of applications that could be using FHE is enormous, as practically any system that requires manipulating personal information. There are both theoretical and practical applications. We can consider FHE as a superset of all other variants presented before, therefore all the other applications are also possible here. As we mentioned before, FHE schemes are not yet very practical. Still, we will show some applications that in the near future, we could be using and applications that researchers or companies have already sampled some use cases.

Zama (HINDI, 2021) is a company that builds open-source code using FHE, mainly for blockchain and artificial intelligence. The team is formed by the creators of (PAILLIER, 1999) and (CHILLOTTI et al., 2020), and they have created several practical examples, showing their advances in FHE. They make simple and practical applications, so the user can check the data is confidential all the time, with a step-by-step system. In the systems, we generate the keys, input some applications-specific information, encrypt it, send it to the server, and then, retrieve the response and decrypt it with our generated private key. Some applications they made using this step-by-step include a sentiment analysis based on some message a user types about his feeling at the moment (the website can be accessed in link), an image filtering on encrypted image (the website is in link), and a health prediction on the fulfillment of some questions about the person’s health (the website is available at link). They made these applications to show how we can combine machine learning training models with FHE, making the user’s data private all

the time.

We have already discussed PIR as an application for PHE and SHE, using simple queries. An actual implementation of total discretion about the user's query can be found in (MENON; WU,) using FHE for complex queries with an organized database, using the PIR protocol in (MENON; WU, 2022). With a database of 6GB of English Wikipedia, the user can search through it and the database manager has no idea what resources were requested. At the first access to the website, it is required to send 18MB of data (public key and other parameters), while later queries require 28KB of upload. The server response is 250KB. This is a very interesting and practical application we can check the slowness and network requirements while using FHE nowadays.

We showed in Section 6.3 that we can make a deep neural network using LFHE. An optimized version using plain FHE, with bootstrap and no need to calculate complex parameters in advance, can be checked in (LEE et al., 2022). They used the fully homomorphic version of CKKS for image classification and achieved 92.43% of accuracy. The problem is the time it takes to infer one image: 3 hours.

Another improvement on some previous achievable applications from HE is MPC. Previously we only considered the gathering of data calculated from each peer, using some additive scheme. Nevertheless, one of the biggest concerns about MPC is about attackers, i.e., how do we know if some party is reliable or not? There are some constructions we can make to overcome this situation, and using FHE now is possible, as shown in (SMART, 2023). They use the fully homomorphic version of TFHE to make their construction, which is secure using a safe oracle model, and efficient to the point of being able to be used in real-life scenarios.

In (MARCOLLA et al., 2022), they present the usage of FHE in data aggregation for fog computing, considering the context of smart cities. Fog computing is a decentralized infrastructure that gathers and process information from devices from the edge of the network, process and transfer it to cloud computing (dedicated servers to store and process huge amount of data). The devices used in fog computing must be event-driven (the device from the edge triggers) and the processing must be packet-by-packet. Thus, it is delay-intolerant (must be performed quickly) and the scope of the processing task is limited to the information contained in a single packet. In this case, citizens' devices could send encrypted data to the fog infrastructure, so no external device would know the actual information. Some problems with this case rely on the speed to process each operation in FHE, the computational complexity, and the possible ciphertext expansion (depending on what scheme would be used). There is also the problem of limited hardware capabilities.

In (SEN, 2013), there are more examples where we could be using FHE, such as:

Protection of mobile agents Homomorphic rings can lead to extensions of $\mathbb{F}_2$, and computer architectures are based on binary strings, requiring only addition and multiplication. This would offer the possibility to encrypt a whole program, so that it is still executable, protecting against malicious hosts;

Oblivious transfer A protocol where the sender transmits information to the receiver, but it stays oblivious, i.e., it does not know exactly what the receiver has actually received. It can be applied to notions of PIR and MPC, where the senders are only a channel for transferring information, but not be able to access that data;

Commitment schemes As a cryptographic primitive where a player makes a commitment, where it is able to choose a value and can no longer change its mind. With FHE, the player would not be demanded to reveal his choice at any moment.

Lottery protocols In a cryptography lottery, a winning number has to be randomly chosen by all participants. Each player chooses a random number which each encrypts, then it can sum the random values and can be efficiently computed. This must use multi-key evaluations to remain secure, so does not have a "moderator" to get all the information and encrypt with only one key;

Mix-nets Protocols that provide anonymity for senders, by collecting encrypted messages from several users and sending them as a ballot. The final user will receive the information but does not know where in the ballot is the information and, thus, it will not know anything about the sender. A similar protocol of mixer is used in the crypto coin Monero, to keep the sender of the money hidden.

Microsoft with its Seal implementation is trying to achieve some applications with FHE (RESEARCH REDMOND, 2023). One of the already implemented applications is password monitoring in their famous browser Edge (LAUTER et al., 2021). This feature compares the ciphertext of users' passwords through many websites and compares it with large password leakage databases. All the passwords are encrypted using an FHE scheme, using their own library Seal. The goal of this password monitoring is to identify weak passwords and alert the user that such a password has already leaked.

7 PRACTICAL IMPLEMENTATION

As part of our work, it was implemented a PHE scheme and an interesting application for PHE in practice. The scheme chosen to be implemented was Okamoto-Uchiyama (OKAMOTO; UCHIYAMA, 1998), which was already studied and shown in Subsection 3.1.7. This scheme was chosen due to its simplicity and efficiency. We also wanted to apply this scheme to our application implementation. The application chosen was the PIR protocol. For that protocol, we only needed an additive PHE scheme that also supports multiplication by scalars/clear texts. Such schemes can be easily seen in Table 1, where Okamoto-Uchiyama is one of them. We recall that the PIR protocol allows a user to retrieve information from a server handling a database, not revealing any information about which data was retrieved.

Before we explain what was performed as a practical implementation for our theme, it is important to show how such a protocol can be achieved with only an additive PHE scheme. Consider a database with only one table that has one column with an integer as its value. Consider one user who wants to access the i-th element to that database, without the *backend* or the database owner to be aware of what i is. For that, we will use the following PIR protocol using PHE, where the user tries to get the i-th element from a database through a server (a visual representation of that description is given in Figure 8):

1. Consider a database d with n elements. The user encrypts the index i of the desired result, using an additive PHE scheme, that supports the multiplication of clear texts. This procedure is done by one-hot encoding encryption, where the index is represented as a vector of encrypted zeros, except for the i-th index, which is the encryption of one. This vector must be of size n, where only the i-th element is the encryption of one. Call this encrypted vector of a.

2. The user sends a to the server.

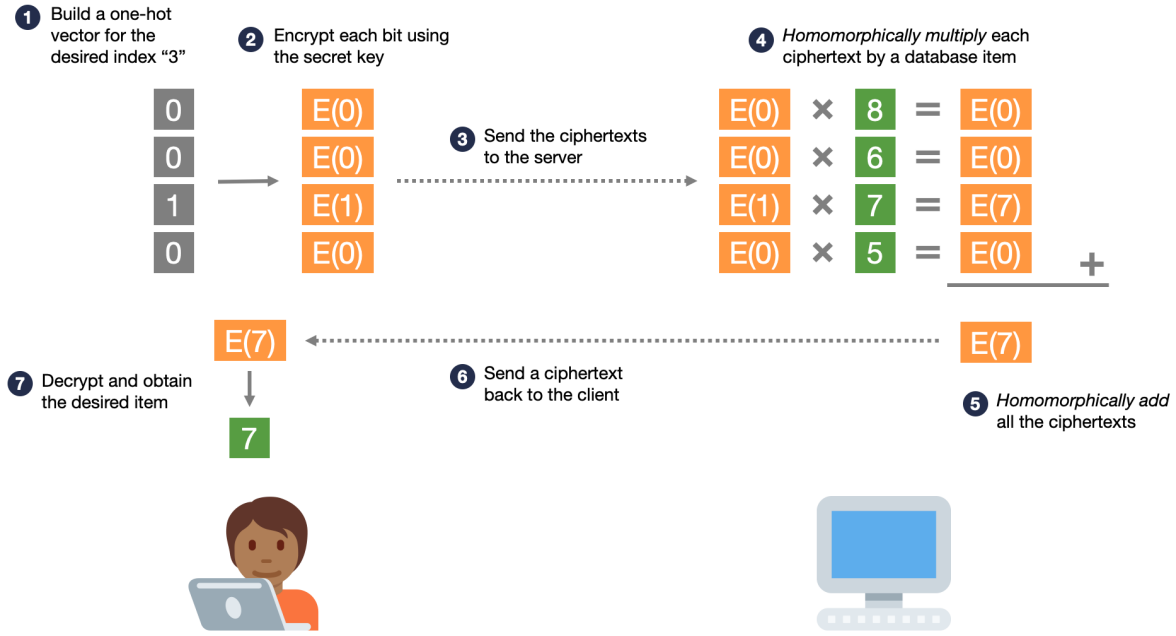
3. The server runs through all elements of d and for each non-ciphered element d_j of index j, it multiplies with the corresponding a_j element. Here we use the multiplication between a ciphertext and a clear text. Then, we need to sum all of the multiplications, homomorphically, which we call s. In other words, $s = \Sigma_{j=1}^{n} d_j \times a_j$.

4. The server sends s back to the client.

5. The client decrypts s, and gets the desired element.

As it can be seen, it is an easy but powerful protocol. Some efficiency problems here involve the necessity of encrypting a vector with the same size as the database and needing to run through all the elements in the database. However, we are only considering a PHE scheme, where there are some limitations. Perhaps for large databases, it is unfeasible, but for smaller databases, it can be considered a great choice.

Figure 8 – Visual representation of the PIR protocol using an additive PHE scheme. Here it asks for the third element in the database, where $n = 4$ and $i = 3$.



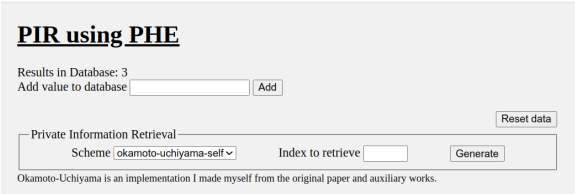
Source: (TEAM, 2022)

Now that we are aware of how the protocol works, we can share our implementation. We made a web application, where the user can insert elements into a database. In order to use the protocol, the user can choose between three different PHE encryption schemes and input the index it wants to retrieve the information. Then, the application will guide to the aforementioned steps, showing details like the public and private key details, the one-hot encoding in clear text and the corresponding encrypted vector, and how much time each operation costs. The idea is to provide an easy-to-use web application, where the PIR protocol is easily understood and where we can check encryption facts visually.

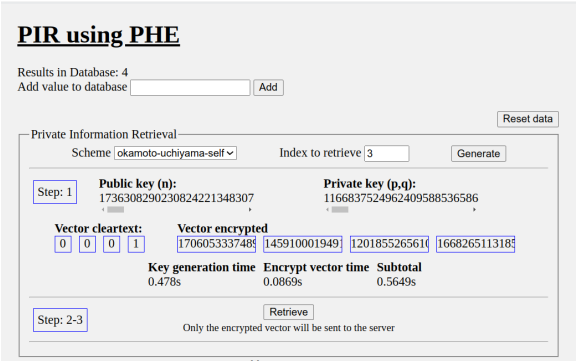
The project is on GitHub at https://github.com/AnthonyKamers/phe-pir and has an MIT license, which means any other person can take this project and use it as they wish, for any means. For the web application, it was used Python 3.10.6, and the *Flask* framework (version 3.0.0) to provide a web client-server. Templates in order to show to the user were made using *Jinja2* (version 3.1.2), which Flask provides easy integration. For the database handling, it was used a very simple table in SQLite, managed by the *peewee* library. The encryption schemes can be chosen between three options: Paillier, Okamoto-Uchiyama, and Damgard-Jurik. The former is managed using the *phe* library (version 1.5.0), while the last was managed using the *damgard-jurik* library (version 0.0.3). As said previously, the Okamoto-Uchiyama was implemented for this work, using the original work as reference (OKAMOTO; UCHIYAMA, 1998), as soon as other auxiliary works to provide improvements in some calculations (KOÇ; ÖZDEMIR; ÖZGER, 2021). We consider the following bit sizes for each scheme: Okamoto-Uchiyama and Paillier: 512 bits; Damgard-Jurik: 64 bits with 3 thresholds.

In order to provide an example of the running application, we will run an example in it. Figure 9a shows the initial state of the web application. In the example, there are already 3 populated numbers in the database. We will add the number 250 in the database now, in order to get 4 elements. For that procedure, it is necessary to put the value 250 into the input text next to its label and click the button “Add”. Now, we have four elements in the database, where we know the 4-th element.

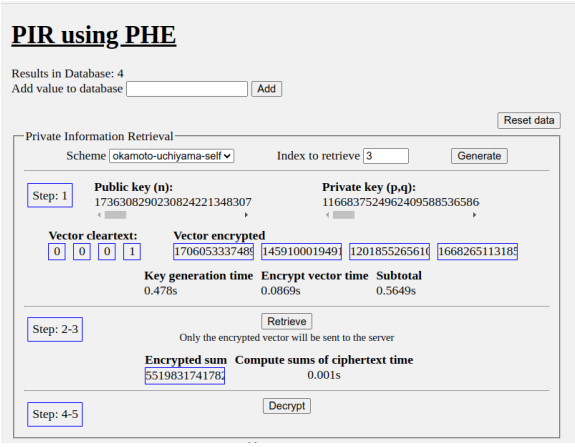
(a) Web application initial state, already populated with 3 random numbers.



(b) Web application in step 1 of the PIR protocol, wanting to know the index 3 of the database.



(c) Web application in steps 2-3 of the PIR protocol.



(d) Complete PIR protocol steps

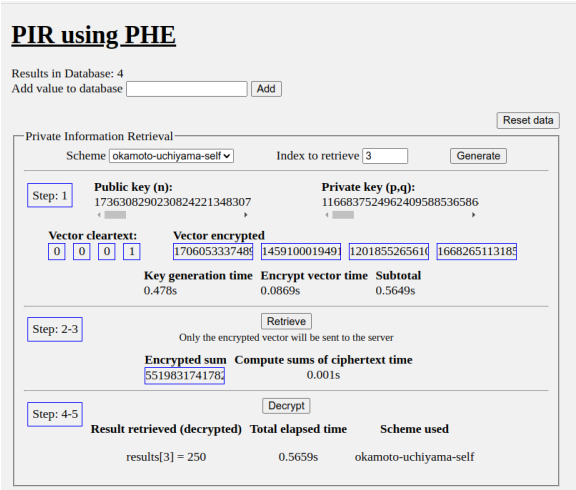


Figure 9 – Web application implemented

Source: the author.

To continue demonstrating the behavior of the PIR protocol, let us select the implemented scheme (*okamoto-uchiyama-self*) and put the value 3 (since in Python the index always starts from 0) in the input text, next to the label “Index to retrieve” and click the button “Generate”. The result will be something similar to Figure 9b, since each time it is generated new random numbers, and the encryptions are probabilistic. In this step, we generate the keys, where it is possible to check the public value n, and the private values p, q. As it is only a system for educational purposes, we expose sensitive information. The one-hot encoding for index 3 is provided, as soon as its corresponding encrypted vector. Also, performance details are given, which will be discussed later.

Now, in order to advance to steps 2-3, we click the button labeled “Retrieve”, where the server will take only the encrypted vector and make the clear text multiplications, and the sum of them homomorphically. The result is shown in Figure 9c, where only the encrypted sum value is shown, and the corresponding time to generate it. The only remaining steps are 4-5, where we need to decrypt the value. Only by clicking the button “Decrypt”, we get the result in Figure 9d. Now, we got the value we wanted, i.e., it is showing $results[3] = 250$, i.e., that the index 3 of “results” (database) is equal to 250. As this was the value we inserted before, we can attest the efficiency of the PIR protocol, using only an additive PHE scheme with support for scalar multiplications.

7.1 PHE SCHEME IMPLEMENTATION AND COMPARISON

As part of our work, we implemented a PHE scheme from those already studied and shown in Chapter 3. The aim of this was to understand all the steps and functions in order to make a PHE cryptosystem work. For that, we chose the Okamoto-Uchiyama scheme. Our focus here is not to show all the functions again, as it was already shown in Subsection 3.1.7. Here, we focus on what was implemented differently from what was explained before and show why.

We recall that for the Okamoto-Uchiyama scheme, we have a public key (n, g, h, k) and a private key (p, q). Our key generation algorithm implemented does not consider the h element, due to performance reasons. So our public key is (n, g, k). We remind that the encryption is given, for a random $r \in \mathbb{Z}/n\mathbb{Z}$ as $\text{ENC}(m) = g^m \times h^r \mod n$. In our implementation, it was very costly to perform the computations of g^m and h^r in separate calculations, taking more than 20 seconds to reach it. Instead, we concatenate them in only one calculation. So, our encryption is given as: $\text{ENC}(m) = g^{m+(n \times r)} \mod n$. It is the same procedure, but uses more parallel calculations in the processor, reaching a plausible time for this computation.

Another interesting point about our implementation is the possibility of running the PIR protocol for multiple PHE schemes and comparing the performance of each step. We have statistics for three cases: key generation, vector encryption, and the multiplication of clear texts and consequently sum of them. One *endpoint* provided from our application, refers to a performance test between the three encryption schemes available: our Okamoto-Uchiyama implementation, Paillier using an external library, and Damgard-Jurik also using an external library.

If the user accesses the endpoint *WebApplicationContext/performance*, where *WebApplicationContext* is where the web application is running, a performance test for 30 iterations will be made for each scheme. In the end, the service will print in the terminal running the web application, the average for each aforementioned case. The values for one call to this endpoint were abstracted here in Table 7. The table is ordered by the total average time between the three different steps. It is possible to check the disparity between the implementation from the libraries, and the one made from this work.

The key generation step is the most disparate, considering our Okamoto-Uchiyama

Table 7 – Comparison of encryption schemes for the PIR web application implemented

Scheme	Key generation	Vector encryption	Sum encrypted	Total
Damgard-Jurik	0.0029	0.0002	0.0001	0.0031
Paillier	0.0021	0.0015	0.0003	0,0039
Okamoto-Uchiyama self	0.5117	0.0972	0.0009	0.6098

All the values are in seconds and are represented as the average after 30 iterations.

implementation. This fact is due mostly to our prime number generation function, where we implemented a random prime generator, using the Miller-Rabin algorithm (RABIN, 1980). First, we generate a number with k bits (in this case, 512) and check if it is a valid prime number by the Miller-Rabin primality test. We repeat this procedure for p and q, until $\gcd(p, q-1) = 1$ and $\gcd(p-1, q) = 1$. This process, as it was performed, is very costly. Nevertheless, the purpose of the implementation of the PHE scheme was to fully understand the whole process of a HE cryptosystem. Performance was not considered a worry for this implementation. We followed the original paper instructions (OKAMOTO; UCHIYAMA, 1998), in order to make it work and some small performance issues were fixed, as explained above.

8 FINAL REMARKS

In this work, we studied the basic principles of encryption in general, focusing the attention on homomorphic encryption. In this type of encryption, we presented the concept, the reason for using it in real-world applications, the different variants, and several applications. We also showed how Gentry was able to make the first fully homomorphic scheme, while researchers spent more than 30 years looking for that. Regarding this variant, we presented the noise problem while computing on ciphertexts, and how Gentry's bootstrap technique can surpass that. Several hard mathematical problems were also explained, to understand the difficulty of an attacker in retrieving someone's ciphertext.

As partially homomorphic and somewhat homomorphic schemes are much more reliable nowadays, we focused on them. We presented an evolution of schemes from its beginning, until the most recent approaches. We showed a detailed explanation of many schemes from these variants, as well as their basic functions and their homomorphic properties. A summary of each variant's schemes was presented, where we were able to easily find the underlying problem regarding scheme security, the respective homomorphic property, and some additional information.

Also, we gave some examples of theoretical and practical applications that we can perform using HE. Some examples were not done in real life yet, due to performance reasons, but it was explained how we could perform them. Also, already existing applications for each variant were given. Additionally, we provided an implementation of a PHE scheme based on the work of Okamoto-Uchiyama, which is additive homomorphic and can perform additions and multiplications over clear texts. We also implemented a practical application where the implemented scheme plays an important role. We chose to implement a private information retrieval protocol using only additive schemes. It is possible to choose among three different schemes to encrypt a query, with our implementation being one of them. A comparison between our implementation and the other schemes was given.

With that in mind, we could check how homomorphic encryption could help improve the privacy of people's data around the world. With the advance of Cloud computing, this is even more necessary, since external companies need to have access to our private key to make modifications to the original data. We gave examples of how we can apply some techniques and make some usages with today's computational power, especially when it comes to applications using a huge amount of additions and a small number of multiplications. PHE and SHE schemes are based on similar underlying problems to the ones that are already being used nowadays, and they have similar performance. This means that, if an application could be using of some technique to be applying HE to protect people's information, this should be done.

We mentioned the difficulty when it comes to processing large amounts of data using more advanced HE variants, such as LFHE and FHE. However, we also presented some very complex applications that handle multiple transformations and can use them nowadays. Applications using machine learning could migrate to HE, so all information about their users would be

preserved, keeping the same prediction accuracy. The sharing of information about HE must be spread throughout the world, so developers know the importance of this strategy and how it completely protects the information someone provides. Some companies are already struggling to add some more functionalities so others can use it more easily, apply it to each time more applications, and have faster performance. The process of migrating from classic cryptography to homomorphic encryption will take a long time, but it is necessary to protect the world's privacy.

8.1 FUTURE WORKS

As we only demonstrated more specific details about PHE and SHE, a more general work, embracing LFHE and FHE schemes' explanations could be done. Besides that, many applications that can be performed with HE, considering any variant, have not yet been proposed or do not have open-source code. A practical implementation of some of these protocols or algorithms would be a great future work, enriching the knowledge about homomorphic encryption, and making the data of users secure and safe, even though it is manipulated by other entities.

BIBLIOGRAPHY

ACAR, A. et al. A survey on homomorphic encryption schemes: Theory and implementation. **ACM Computing Surveys (Csur)**, ACM New York, NY, USA, v. 51, n. 4, p. 1–35, 2018.

ALAGIC, G. et al. Quantum fully homomorphic encryption with verification. In: SPRINGER. **International Conference on the Theory and Application of Cryptology and Information Security**. [S.l.], 2017. p. 438–467.

ALBRECHT, M.; BAI, S.; DUCAS, L. A subfield lattice attack on overstretched ntru assumptions: Cryptanalysis of some fhe and graded encoding schemes. In: SPRINGER. **Annual International Cryptology Conference**. [S.l.], 2016. p. 153–178.

ALBRECHT, M. et al. **Homomorphic Encryption Security Standard**. Toronto, Canada, 2018.

ALBRECHT, M. R. et al. Polly cracker, revisited. In: SPRINGER. **International Conference on the Theory and Application of Cryptology and Information Security**. [S.l.], 2011. p. 179–196.

ALWEN, J. **What is lattice-based Cryptography & why you should care**. Wickr Cryptography, 2020. Disponível em: https://medium.com/cryptoblog/what-is-lattice-based-cryptography-why-should-you-care-dbf9957ab717.

AMORIM, I.; COSTA, I. Leveraging searchable encryption through homomorphic encryption: A comprehensive analysis. **Mathematics**, MDPI, v. 11, n. 13, p. 2948, 2023.

ARMKNECHT, F.; SADEGHI, A.-R. A new approach for algebraically homomorphic encryption. **Cryptology ePrint Archive**, 2008.

BARKEE, B. et al. Why you cannot even hope to use gröbner bases in public key cryptography: an open letter to a scientist who failed and a challenge to those who have not yet failed. **Journal of Symbolic Computation**, Elsevier, v. 18, n. 6, p. 497–501, 1994.

BENALOH, J. Dense probabilistic encryption. In: **Proceedings of the workshop on selected areas of cryptography**. [S.l.: s.n.], 1994. p. 120–128.

BONEH, D.; GOH, E.-J.; NISSIM, K. Evaluating 2-dnf formulas on ciphertexts. In: SPRINGER. **Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2**. [S.l.], 2005. p. 325–341.

BRAKERSKI, Z.; GENTRY, C.; VAIKUNTANATHAN, V. (leveled) fully homomorphic encryption without bootstrapping. **ACM Transactions on Computation Theory (TOCT)**, ACM New York, NY, USA, v. 6, n. 3, p. 1–36, 2014.

BRAKERSKI, Z.; VAIKUNTANATHAN, V. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: SPRINGER. **Annual cryptology conference**. [S.l.], 2011. p. 505–524.

BRIGHT, C. From the shortest vector problem to the dihedral hidden subgroup problem. In: . [S.l.: s.n.], 2013.

BUCHANAN, W. J. **Learning With Errors (LWE) and Ring LWE**. 2023. Accessed: November 12, 2023. Disponível em: https://asecuritysite.com/encryption/lwe-output.

CHATTERJEE, A.; SENGUPTA, I. Translating algorithms to handle fully homomorphic encrypted data on the cloud. **IEEE Transactions on Cloud Computing**, IEEE, v. 6, n. 1, p. 287–300, 2015.

CHEON, J. H.; JEONG, J.; LEE, C. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero. **LMS Journal of Computation and Mathematics**, London Mathematical Society, v. 19, n. A, p. 255–266, 2016.

CHEON, J. H. et al. Homomorphic encryption for arithmetic of approximate numbers. In: SPRINGER. **International conference on the theory and application of cryptology and information security**. [S.l.], 2017. p. 409–437.

CHEON, J. H.; STEHLÉ, D. Fully homomophic encryption over the integers revisited. In: SPRINGER. **Annual International Conference on the Theory and Applications of Cryptographic Techniques**. [S.l.], 2015. p. 513–536.

CHILLOTI, I. Introduction to fhe and the tfhe scheme. In: . [S.l.]: Workshop on Foundations and Applications of Lattice-based Cryptography ICMS, Edinburgh, 2022.

CHILLOTTI, I. et al. Tfhe: fast fully homomorphic encryption over the torus. **Journal of Cryptology**, Springer, v. 33, n. 1, p. 34–91, 2020.

CRAMER, R.; DAMGÅRD, I.; NIELSEN, J. B. Multiparty computation from threshold homomorphic encryption. In: SPRINGER. **Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20**. [S.l.], 2001. p. 280–300.

CRAMER, R.; GENNARO, R.; SCHOENMAKERS, B. A secure and optimally efficient multi-authority election scheme. **European transactions on Telecommunications**, Wiley Online Library, v. 8, n. 5, p. 481–490, 1997.

DAMGÅRD, I.; JURIK, M. A length-flexible threshold cryptosystem with applications. In: SPRINGER. **Information Security and Privacy: 8th Australasian Conference, ACISP 2003 Wollongong, Australia, July 9–11, 2003 Proceedings 8**. [S.l.], 2003. p. 350–364.

DIFFIE, W. New direction in cryptography. **IEEE Trans. Inform. Theory**, v. 22, p. 472–492, 1976.

DIJK, M. v. et al. Fully homomorphic encryption over the integers. In: SPRINGER. **Annual international conference on the theory and applications of cryptographic techniques**. [S.l.], 2010. p. 24–43.

DORÖZ, Y.; SUNAR, B. Flattening ntru for evaluation key free homomorphic encryption. **Journal of Mathematical Cryptology**, De Gruyter, v. 14, n. 1, p. 66–83, 2020.

DOWERAH, U.; KRISHNASWAMY, S. Fully homomorphic encryption based on multivariate polynomial evaluation. **arXiv preprint arXiv:1910.06270**, 2019.

DUCAS, L.; MICCIANCIO, D. Fhew: bootstrapping homomorphic encryption in less than a second. In: SPRINGER. **Annual international conference on the theory and applications of cryptographic techniques**. [S.l.], 2015. p. 617–640.

ELGAMAL, T. A public key cryptosystem and a signature scheme based on discrete logarithms. **IEEE transactions on information theory**, IEEE, v. 31, n. 4, p. 469–472, 1985.

FAN, J.; VERCAUTEREN, F. Somewhat practical fully homomorphic encryption. **Cryptology ePrint Archive**, 2012.

FELLOWS, M.; KOBLITZ, N. Combinatorial cryptosystems galore! **Contemporary Mathematics**, American Mathematical Society, v. 168, p. 51–51, 1994.

FULLY Homomorphic Encryption. IBM. https://research.ibm.com/topics/fully-homomorphic-encryption. Disponível em: https://research.ibm.com/topics/fully-homomorphic-encryption.

GENTRY, C. **A fully homomorphic encryption scheme**. [S.l.]: Stanford university, 2009.

GENTRY, C.; SAHAI, A.; WATERS, B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: SPRINGER. **Annual Cryptology Conference**. [S.l.], 2013. p. 75–92.

GOLDWASSER, S.; MICALI, S. Probabilistic encryption & how to play mental poker keeping secret all partial information. In: **Proceedings of the fourteenth annual ACM symposium on Theory of computing**. [S.l.: s.n.], 1982. p. 365–377.

GUEVARA, M. **Expanding our Fully Homomorphic Encryption offering**. Google, 2023. https://developers.googleblog.com/2023/08/expanding-our-fully-homomorphic-encryption-offering.html. Disponível em: https://developers.googleblog.com/2023/08/expanding-our-fully-homomorphic-encryption-offering.html.

HARJITO, B. et al. Comparative analysis of rsa and ntru algorithms and implementation in the cloud. **International Journal of Advanced Computer Science and Applications**, Science and Information (SAI) Organization Limited, v. 13, n. 3, 2022.

HEROLD, G. Polly cracker, revisited, revisited. In: SPRINGER. **Public Key Cryptography–PKC 2012: 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings 15**. [S.l.], 2012. p. 17–33.

HINDI, R. **People shouldn't care about privacy**. Zama AI, 2021. Disponível em: https://www.zama.ai/post/people-should-not-care-about-privacy.

HOVD, M. N. **The handling of noise and security of two fully homomorphic encryption schemes**. Dissertação (Mestrado) — NTNU, 2017.

IBARRONDO, A. et al. Grote: Group testing for privacy-preserving face identification. In: **Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy**. [S.l.: s.n.], 2023. p. 117–128.

ILIASHENKO, I.; NEGRE, C.; ZUCCA, V. Integer functions suitable for homomorphic encryption over finite fields. In: **Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography**. [S.l.: s.n.], 2021. p. 1–10.

ISHAI, Y.; PASKIN, A. Evaluating branching programs on encrypted data. In: SPRINGER. **Theory of Cryptography Conference**. [S.l.], 2007. p. 575–594.

KAWACHI, A.; TANAKA, K.; XAGAWA, K. Multi-bit cryptosystems based on lattice problems. In: SPRINGER. **Public Key Cryptography–PKC 2007: 10th International Conference on Practice and Theory in Public-Key Cryptography Beijing, China, April 16-20, 2007. Proceedings 10**. [S.l.], 2007. p. 315–329.

KIRCHNER, P.; FOUQUE, P.-A. Revisiting lattice attacks on overstretched ntru parameters. In: SPRINGER. **Annual International Conference on the Theory and Applications of Cryptographic Techniques**. [S.l.], 2017. p. 3–26.

KNIRSCH, F. et al. Comparison of the paillier and elgamal cryptosystems for smart grid aggregation protocols. In: **ICISSP**. [S.l.: s.n.], 2020. p. 232–239.

KOBLITZ, N. **A course in number theory and cryptography**. [S.l.]: Springer Science & Business Media, 1994. v. 114.

KOÇ, Ç. K.; ÖZDEMIR, F.; ÖZGER, Z. Ö. **Partially Homomorphic Encryption**. [S.l.]: Springer, 2021.

KUNDRO, D. **Criptografia homomórfica: um esquema de criptografia cada vez mais usado**. [S.l.], 2019. Disponível em: https://www.welivesecurity.com/br/2019/09/06/criptografia-homomorfica-um-esquema-de-criptografia-cada-vez-mais-usado/. Acesso em: 18 jul. de 2022.

LANGLOIS, A.; STEHLÉ, D. Worst-case to average-case reductions for module lattices. **Designs, Codes and Cryptography**, Springer, v. 75, n. 3, p. 565–599, 2015.

LAUTER, K. et al. **Password monitor: Safeguarding passwords in microsoft edge**. Microsoft, 2021. Disponível em: https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/.

LEE, J.-W. et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. **IEEE Access**, IEEE, v. 10, p. 30039–30054, 2022.

LITTLE, B. **6 World War II Inovations That Changed Everyday Life**. [S.l.], 2021. Disponível em: https://www.history.com/news/world-war-ii-innovations. Acesso em: 16 nov. de 2022.

LÓPEZ-ALT, A.; TROMER, E.; VAIKUNTANATHAN, V. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: **Proceedings of the forty-fourth annual ACM symposium on Theory of computing**. [S.l.: s.n.], 2012. p. 1219–1234.

LOU, Q.; JIANG, L. She: A fast and accurate deep neural network for encrypted data. **Advances in neural information processing systems**, v. 32, 2019.

LY, L. V. Polly two: A new algebraic polynomial-based public-key scheme. **Applicable Algebra in Engineering, Communication and Computing**, Springer, v. 17, n. 3-4, p. 267–283, 2006.

LYUBASHEVSKY, V.; PEIKERT, C.; REGEV, O. On ideal lattices and learning with errors over rings. In: SPRINGER. **Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29**. [S.l.], 2010. p. 1–23.

MARCOLLA, C. et al. Survey on fully homomorphic encryption, theory, and applications. **Proceedings of the IEEE**, IEEE, v. 110, n. 10, p. 1572–1609, 2022.

MARTINS, P.; SOUSA, L.; MARIANO, A. A survey on fully homomorphic encryption: An engineering perspective. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 50, n. 6, p. 1–33, 2017.

MELCHOR, C. A.; GABORIT, P.; HERRANZ, J. Additively homomorphic encryption with d-operand multiplications. In: SPRINGER. **Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30**. [S.l.], 2010. p. 138–154.

MENON, S. J.; WU, D. J. **Spiral demo**. IACR. Disponível em: https://spiralwiki.com/.

MENON, S. J.; WU, D. J. Spiral: Fast, high-rate single-server pir via fhe composition. In: IEEE. **2022 IEEE Symposium on Security and Privacy (SP)**. [S.l.], 2022. p. 930–947.

MITTAL, S.; RAMKUMAR, K. A retrospective study on ntru cryptosystem. In: AIP PUBLISHING. **AIP Conference Proceedings**. [S.l.], 2022. v. 2451, n. 1.

NACCACHE, D.; STERN, J. A new public key cryptosystem based on higher residues. In: **Proceedings of the 5th ACM Conference on Computer and Communications Security**. [S.l.: s.n.], 1998. p. 59–66.

NAEHRIG, M.; LAUTER, K.; VAIKUNTANATHAN, V. Can homomorphic encryption be practical? In: **Proceedings of the 3rd ACM workshop on Cloud computing security workshop**. [S.l.: s.n.], 2011. p. 113–124.

OKAMOTO, T.; UCHIYAMA, S. A new public-key cryptosystem as secure as factoring. In: SPRINGER. **Advances in Cryptology—EUROCRYPT'98: International Conference on the Theory and Application of Cryptographic Techniques Espoo, Finland, May 31–June 4, 1998 Proceedings 17**. [S.l.], 1998. p. 308–318.

PAILLIER, P. Public-key cryptosystems based on composite degree residuosity classes. In: STERN, J. (Ed.). **Advances in Cryptology — EUROCRYPT '99**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. p. 223–238.

PARK, J.; TIBOUCHI, M. Shecs-pir: somewhat homomorphic encryption-based compact and scalable private information retrieval. In: SPRINGER. **European Symposium on Research in Computer Security**. [S.l.], 2020. p. 86–106.

PATERSON, M. S.; STOCKMEYER, L. J. On the number of nonscalar multiplications necessary to evaluate polynomials. **SIAM Journal on Computing**, SIAM, v. 2, n. 1, p. 60–66, 1973.

PFITZMANN, B.; SCHUNTER, M. Asymmetric fingerprinting. In: SPRINGER. **International Conference on the Theory and Applications of Cryptographic Techniques**. [S.l.], 1996. p. 84–95.

RABIN, M. O. Probabilistic algorithm for testing primality. **Journal of number theory**, Elsevier, v. 12, n. 1, p. 128–138, 1980.

REGEV, O. The learning with errors problem. **Invited survey in CCC**, v. 7, n. 30, p. 11, 2010.

RESEARCH REDMOND, W. M. **Microsoft SEAL (release 4.1)**. 2023. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Disponível em: https://github.com/Microsoft/SEAL.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, ACM New York, NY, USA, v. 21, n. 2, p. 120–126, 1978.

ROCHA, V.; LÓPEZ, J.; ROCHA, V. F. D. An overview on homomorphic encryption algorithms. **UNICAMP Universidade Estadual de Campinas, Tech. Rep**, 2018.

SANDER, T.; YOUNG, A.; YUNG, M. Non-interactive cryptocomputing for nc/sup 1. In: IEEE. **40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)**. [S.l.], 1999. p. 554–566.

SAVIĆ, D. et al. An application of partial homomorphic encryption in computer system with limited resources. **Tehnički vjesnik**, Strojarski fakultet u Slavonskom Brodu; Fakultet elektrotehnike, računarstva . . . , v. 25, n. 3, p. 709–713, 2018.

SEN, J. Homomorphic encryption — theory and application. In: SEN, J. (Ed.). **Theory and Practice of Cryptography and Network Security Protocols and Technologies**. Rijeka: IntechOpen, 2013. cap. 1. Disponível em: https://doi.org/10.5772/56687.

SHAMIR, A. How to share a secret. **Communications of the ACM**, ACm New York, NY, USA, v. 22, n. 11, p. 612–613, 1979.

SHOR, P. W. Algorithms for quantum computation: discrete logarithms and factoring. In: IEEE. **Proceedings 35th annual symposium on foundations of computer science**. [S.l.], 1994. p. 124–134.

SILVA, E. A. da. Practical use of partially homomorphic cryptography. 2016.

SMART, N. P. Practical and efficient fhe-based mpc. **Cryptology ePrint Archive**, 2023.

SMART, N. P.; VERCAUTEREN, F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In: SPRINGER. **International Workshop on Public Key Cryptography**. [S.l.], 2010. p. 420–443.

SMART, N. P.; VERCAUTEREN, F. Fully homomorphic simd operations. **Designs, codes and cryptography**, Springer, v. 71, p. 57–81, 2014.

TEAM, B. **Private information retrieval using homomorphic encryption (explained from scratch)**. Blyss dev, 2022. Disponível em: https://blintzbase.com/posts/pir-and-fhe-from-scratch/.

WILLIAMS, V. V. Multiplying matrices faster than coppersmith-winograd. In: **Proceedings of the forty-fourth annual ACM symposium on Theory of computing**. [S.l.: s.n.], 2012. p. 887–898.

WOODY, H. Polynomial resultants. **GNU operating system**, 2016.

Yackel, Ryan. **What is homomorphic encryption?** [S.l.], 2021. Disponível em: https://www.keyfactor.com/blog/what-is-homomorphic-encryption/. Acesso em: 20 nov. de 2022.

APPENDIX A – ARTIGO DO TCC

Homomorphic encryption
Introduction and Applicabilities

Kamers, Anthony Bernardo[1]

[1]Departamento de Informática e Estatística
Universidade Federal De Santa Catarina (UFSC)
Florianópolis – SC – 88040-900 – Brazil

anthony.kamers@grad.ufsc.br

Abstract. *Since the invention of the RSA cryptosystem in 1978 [Rivest et al. 1978], it was imagined a world where we could operate only in encrypted data. RSA itself is homomorphic by multiplication, meaning we can make multiplications of ciphertexts, and still be able to decrypt it correctly. This paper gives an introduction to homomorphic encryption, the difficulties involved in the process of making such schemes, the different variants of it, and the potential applications for some of the variants.*

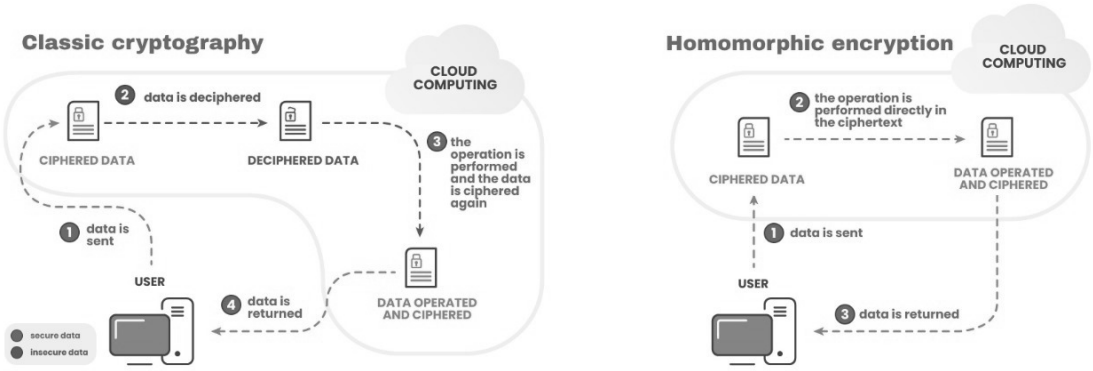
1. Introduction and motivation

Technology is constantly present in people's lives, making it necessary to provide several personal information to computational systems, which will process this data, analyze and execute operations on it. With personal data processing, comes the need for privacy-preserving solutions. The problem is the lack of trust in the processing of this knowledge by third-party applications. Several times newspapers around the world show countless examples of exposition of confidential information. This information is usually transmitted using encryption such as TLS, to guarantee data confidentiality. However in order to analyze this data (in machine learning, for instance), it is necessary to operate it on clear text, needing to decrypt it first. At this moment, the data is vulnerable and susceptible to attacks or leaks of private information (by the cloud server), as will be explained hereafter.

To exemplify how homomorphic encryption could bring security benefits over private user data, we can cite the processing of information by cloud computing. In a scenario where there is no homomorphic encryption, the application receives encrypted data, decrypts it, and then performs over clear text. At this moment, the information is unprotected, being subject to attacks or even, malicious use of the data by the server that is doing the analysis. After the data is processed, the result is encrypted and sent back to the user. Now, if we use homomorphic encryption, there would be no need to decrypt the data at any moment in the flow of the data analysis, since the operations of this type of scheme can be made directly on the encrypted text, letting the data protected and safe all along the way. Figure 1 shows the difference between the two possibilities.

Homomorphic encryption has the potential to solve many security and privacy issues, in a variety of daily applications, such as cloud computing, data science, and zero-knowledge proof, among others [Alagic et al. 2017]. Besides that, it can help applications to operate in conformity with data protection laws, such as the Brazilian LGPD (Lei Geral de Proteção dos Dados) or the European GDPR (General Data Protection Regulation), because user data will not be decrypted.

Figure 1. Flow of information exchanged between servers using and not using homomorphic encryption. Translated from [Kundro 2019]



So, we can observe there are several benefits in using homomorphic encryption, nevertheless, it is important to highlight that there are some limitations on the operations that can be made on encrypted data (as addition and/or multiplication) and the number of times it can execute on (depth). Therefore, different schemes have different properties. There are a few classifications of homomorphic encryption:

- Partially Homomorphic Encryption (PHE): supports only one type of operation, such as addition or multiplication. Does not have a restriction on the number of operations that can be computed;
- Somewhat Homomorphic Encryption (SHE): supports two types of operation, but can be operated only for a specific number of times (the most common case is an unlimited number of additions and a small number, usually one, of multiplications);
- Leveled Fully Homomorphic Encryption (LFHE): allows any type of operation (addition and multiplication) for a pre-established depth (the specific number of times it allows to be made);
- Fully Homomorphic Encryption (FHE): allows any type of operation with unlimited depth. This is the most difficult to achieve mathematically and, also, the one with more applications.

2. Definitions

Some important definitions to handle precise information will be given in the next subsections.

2.1. Homomorphism

Homomorphism is, in mathematics, a mapping between two algebraic structures of the same type, that preserves the operations of these structures. Consider the function f defined as $A \to B$ (being A the function's domain and B the respective image), where A and B are sets equipped with a binary operation $*$. The homomorphism preserves the operation of the structures, meaning $f(a * b) = f(a) * f(b)$ for all elements $a, b \in A$. When we study homomorphic encryption, the function f is the encryption or decryption function, with the operation $*$ depending on the scheme.

2.2. Noise

Noise is a parameter of encryption functions in probabilistic schemes, such as fully homomorphic schemes. It is necessary to apply some randomness to the encryption channel, so only the owner can remove it and obtain the original message. When we "include" some random parameter in cryptographic functions, they end up not being deterministic, but rather probabilistic. To that randomness, we call it error or noise. If the encryption of the same message is always the same ciphertext, then we call it a deterministic scheme.

Now that we know why noise is important to encryption schemes, the question is how this can disturb the decryption over operated ciphertexts. The problem is that "classic" encryption schemes do not change their ciphertext after it is encrypted, but homomorphic ciphertexts are changeable, because of the possible operations over it. This means that, if we operate on the ciphertext too many times, we will not be able to decrypt it correctly. When this happens, we call the ciphertext to get unmanageable.

2.3. Bootstrap

As an old open problem until 2009, the proposal of Gentry called bootstrap [Gentry 2009] was a mark in the world of cryptography. Besides creating an FHE, he also created a method to convert a SHE scheme into an FHE one. As we noticed from the previous subsection, the difficult problem in making an FHE scheme is how to manage the noise growth. His approach is based on the idea of shrinking the noise when it gets too large.

To bootstrap some ciphertext, we need two new elements in the encryption scheme, a public evaluation key (also called bootstrapping key) and a $recrypt$ function. The evaluation key is used in the recryption process. The recrypt function is used to reduce the noise when it gets too large. To do that, it is necessary to re-encrypt the ciphertext with an evaluation key k, and this will reset the noise. This process gives us a ciphertext of a ciphertext. To obtain the ciphertext of the original message x, we use the evaluation key again, which is a public value. Let us take the expression $c = \text{ENC}(x, k)$ as the original ciphertext and $c' = \text{ENC}(c, k')$ as the encryption of the first ciphertext. Using the evaluation key, we can decrypt c' and get the ciphertext of the original clear text.

2.4. Leveled approach

LFHE schemes can be very often related to more practical uses nowadays than the bootstrap approach, due to its speed in performing operations over the ciphertext and decrypting it. This happens because LFHE can only make a limited amount of operations, which the user can specify. If we are aware of the amount of times we need to operate on the ciphertext, we can make some improvements so the noise gets controlled.

This is especially essential when it involves costly operations, such as multiplications. Some schemes like CKKS [Cheon et al. 2017] involve up to three additional steps in the process of multiplying two ciphertexts, for instance. Because of the careful noise management, this HE variant is also related to different public key sizes. The problem is when we need a large number of multiplications, then it is probably better to use an FHE scheme.

3. Applications

As we noticed, LFHE and FHE are very versatile and we could achieve practically any application. The problem is the size of the keys and the time to make operations over

the ciphertext. Until today we do not have schemes from those variants that are fast enough for real-world applications. Nevertheless, PHE and SHE schemes are very fast and reliable until today. Although we can only perform one type of operation (or two using SHE), there are many applications we can use, to fully protect the privacy of people. For that reason, we shall explain some applications only for those two variants.

3.1. Partially Homomorphic Encryption

Applications for PHE schemes are limited, due to having only one type of operation that can be performed over the ciphertext. Nevertheless, some important privacy-preserving applications can make use of that. Some examples are online voting systems, which can calculate the number of votes for each participant in the election by homomorphically adding all the votes; blockchain ledgers, whose only operation needed to know the exact balance or the transaction history is adding the past transactions, which can be made homomorphically; in [Savić et al. 2018], they demonstrate that for some specific IoT areas, such as smart home sensors, it is possible to perform transactions between cloud-computer-sensors using PHE encryption and only call the sensor again after some server calculations and over scalars.

Applications mentioned in [Benaloh 1994] consider verifiable secret sharing and verifiable secret ballot elections. The first can be considered as a way to divide a secret into shares, for different shareholders (where each one holds a secret key). The only way to reconstruct the complete secret is by calculating a specific amount from n shareholders. The authors show that for a determined polynomial, the only operation necessary to reconstruct the secret is by performing the addition of each part of it, in this case, each part of the secret, i.e. the share of each shareholder. It is verifiable, so one dishonest part cannot disrupt the reconstruction of the secret, as it can only be reconstructed if a sufficient amount of trusted shareholders give their own share. The verifiable secret-ballot elections are similar to the idea presented before, where a central entity creates an additive probabilistic PHE and makes the public key, public. In that way, each voter can compute, for a new random value, the encryption of yes (1) or no (0) and put it into the ballot. Then, the central entity, which possesses the private key only needs to sum all the votes and homomorphically decrypt it, revealing the result of the election.

In [Naccache and Stern 1998, Pfitzmann and Schunter 1996], they propose to put a watermark on some value or product. Aiming the software industry or illegal image and music reproduction, it is possible to create a watermark, so that only the buyer knows the data with the fingerprint. Still, if the merchant finds this copy somewhere else, he can identify the buyer and prove that the determined buyer bought the copy and is, perhaps, illegally distributing it. The computations can be made homomorphically and it only needs to perform additions over the ciphertext.

In [Cramer et al. 2001], it is proposed the usage of PHE like Paillier's for MPC. Multiparty computations rely on the multi-processing of data between different peers, while the peers only know pieces of information about the original data. Originally, it was proposed the use of Shamir's secret sharing [Shamir 1979] to divide the information between n peers, in a way that can only be reconstructed together using a specific amount of the shares. However, in MPC we can send one piece to each party, encrypted; then each party performs a calculation; at the end, the encrypted result of all peers is added

together homomorphically and then decrypted. Only the owner of the data possesses the secret key to decrypt it and get the calculation together of the peers.

One other application of PHE is keyword search. The work of [Amorim and Costa 2023] shows that several searchable schemes use raw PHE schemes and they are very efficient. They gathered multiple schemes that can do such computation efficiently and classified what strategies each uses. They also showed that multiple strategies consider PHE to reach keyword searches. However, they can execute only some raw operations such as single-keyword search in sequential scanning for single users. In [da Silva 2016], it is implemented a file search application, using already established homomorphic libraries.

Yet another example is the private information retrieval protocol (PIR), which relies on the problem that, given a public database and a user wanting to query some information on it, it is difficult to hide that query from the database's operator. The protocol defines a way for the user to encrypt the query, and only the user has the public key to decrypt it. There are multiple implementations of such protocol, where it is necessary to iterate the whole database to perform a hidden query. Some of these arrangements can be constructed only by using an additive PHE with support for scalar multiplications.

3.2. Somewhat Homomorphic Encryption

We will give some ideas of projects, protocols, equations, and general applications we could be using SHE. We need to keep in mind that with SHE, we cannot use so many multiplications, because all schemes only support a small number of them. So we need to think of ways to minimize the amount of this operation or substitute them with an equivalent operation; for instance, if the user wants to perform a calculation that performs a low-depth of multiplications, we can substitute them for multiple sums of the number (it is necessary the interaction with the user); alternatively, if the user needs to multiply by a constant, that could not affect the retrieval of whole information, we could use a scheme that allows to do it using multiplication by a scalar.

In [Naehrig et al. 2011], authors consider some systems that require many additions and only a small number of multiplication over ciphertexts. They consider applications in some very important areas throughout the world, like medical, financial, and advertisements. They consider both functions that should operate over sensitive information and functions that are proprietary. They also mention some very important calculations that are used almost anywhere and could be performed only with SHE: average (which requires no multiplication), standard deviation (only one multiplication required), and logistical regression (only a small number of multiplications, depending on the precision required). If we have logistical regression, we can evaluate a wide variety of applications that require Big Data and artificial intelligence. One problem that is also mentioned is the constraint of not knowing how to divide real numbers or take square roots with these schemes. All of the aforementioned applications then should return the encryption of the numerator and the encryption of the denominator, then the user decrypts it and makes the calculation herself.

Although not very practical, it is important to be aware of the existence of such possibilities, so that in the future, we might resolve it somehow. In [Iliashenko et al. 2021], they demonstrate some interesting applications using only SHE

in some cases, such as the parity function (checking if a number is odd or even), if a number is the power of another, taking the modulo of a number, taking the Hamming weight (accounts the number of symbols that are different from zero) and, also, the Hamming distance (number of positions different from two strings), performing the conditional less-than function, and indicate how to operate $c \pmod 2$, being c a ciphertext. These functions can be performed using much fewer multiplications because some techniques are applied involving finite fields theory.

Some more complex applications can be formed, with many more operations and function evaluations. We cannot forget that SHE schemes are a super-set of applications within the context of PHE. So all the applications explained earlier can be combined with these, as a way of making more elaborated utilization. Also, we cannot forget the branching programs introduced by Ishai-Paskin [Ishai and Paskin 2007], which enables us to a variety of domains.

As stated previously in PHE applications (see 3.1), we can make PIR protocols using only PHE. Multiple implementations use homomorphic encryption over whole databases and perform queries over ciphertexts. Still, they are very inefficient and also need to handle the whole database, with the additional overhead of HE for itself. In [Park and Tibouchi 2020] is proposed a small and contained way for PIR protocol, where it is only necessary to iterate $O(\log n)$ items from the database, and it is very compact. They use compressing techniques of ciphertexts in the database and over the user's query using a SHE.

In the work of Ishai-Paskin [Ishai and Paskin 2007], the authors proposed a protocol that enhances the ability of keyword search (as it is already achievable with PHE). With that, it is possible to completely hide from the client the original size of the database, whereas it is also possible to hide from the database operator anything related to the client's query. They can perform these tasks in one round, i.e., it is not necessary to use a three-way handshake as in the majority of other schemes.

4. Final remarks

In this paper, we studied the basic principles of homomorphic encryption and why it should be used instead of classic encryption. We explained why it is superior in cloud computing environments, giving examples and comparing it to classic cryptography. It was given an introduction to the "noise" problem, explaining why it is hard to achieve FHE. We explained the different variants of homomorphic encryption and focused on the possible applications to the "basic" variants: PHE and SHE. It is possible to use schemes from these variants nowadays, due to their speed and small key sizes.

References

Alagic, G., Dulek, Y., Schaffner, C., and Speelman, F. (2017). Quantum fully homomorphic encryption with verification. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 438–467. Springer.

Amorim, I. and Costa, I. (2023). Leveraging searchable encryption through homomorphic encryption: A comprehensive analysis. *Mathematics*, 11(13):2948.

Benaloh, J. (1994). Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography*, pages 120–128.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pages 409–437. Springer.

Cramer, R., Damgård, I., and Nielsen, J. B. (2001). Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*, pages 280–300. Springer.

da Silva, E. A. (2016). Practical use of partially homomorphic cryptography.

Gentry, C. (2009). *A fully homomorphic encryption scheme*. Stanford university.

Iliashenko, I., Negre, C., and Zucca, V. (2021). Integer functions suitable for homomorphic encryption over finite fields. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 1–10.

Ishai, Y. and Paskin, A. (2007). Evaluating branching programs on encrypted data. In *Theory of Cryptography Conference*, pages 575–594. Springer.

Kundro, D. (2019). Criptografia homomórfica: um esquema de criptografia cada vez mais usado. Technical report.

Naccache, D. and Stern, J. (1998). A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 59–66.

Naehrig, M., Lauter, K., and Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124.

Park, J. and Tibouchi, M. (2020). Shecs-pir: somewhat homomorphic encryption-based compact and scalable private information retrieval. In *European Symposium on Research in Computer Security*, pages 86–106. Springer.

Pfitzmann, B. and Schunter, M. (1996). Asymmetric fingerprinting. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 84–95. Springer.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Savić, D., Trikoš, M., Veinović, M., and Simić, D. (2018). An application of partial homomorphic encryption in computer system with limited resources. *Tehnički vjesnik*, 25(3):709–713.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.