Fully Homomorphic Encryption and its Use Cases

Felix Peter Paul
Johannes Gutenberg University Mainz
Mainz, Rhineland-Palatinate, Germany
fepaul@students.uni-mainz.de

ABSTRACT

Fully Homomorphic Encryption (FHE) is a groundbreaking cryptographic technique that enables computation on encrypted data, maintaining confidentiality even during processing. This technology is increasingly relevant due to stringent privacy laws like the General Data Protection Regulation (GDPR) and the rise of significant and costly data breaches in cloud environments. It holds immense potential in many sectors such as healthcare, for secure analysis of encrypted patient records, or finance, for confidential processing of financial data in unregulated environments.

This paper aims to provide foundational knowledge on FHE, position it within the field of cryptography, review key schemes and implementations, and specifically explore its practical use cases.

1 INTRODUCTION

Fully Homomorphic Encryption (FHE) stands as a pivotal innovation in the realm of cryptography, hailed as the 'holy grail' [22] for its ability to process any function on encrypted data, without ever needing to decrypt them into plaintexts. Traditional encryption techniques only allow encryption at rest. FHE on the other hand allows encryption at processing, thereby preventing data extractions throughout the computation process. The significance of FHE is magnified in an era where data breaches and privacy concerns are rampant, which exposed the personal information of millions of individuals. and where legislation such as the European Union's General Data Protection Regulation (GDPR) mandates stringent data protection measures.

The concept of FHE was first introduced in 1978 by Rivest, Adleman, and Dertouzos, who envisioned the idea of 'privacy homomorphisms' – a precursor to what we now understand as FHE [24]. While RSA, a widely-known cryptographic system developed in 1977 by Rivest, Shamir, and Adleman, is often cited as an early example of a scheme supporting a single homomorphic operation, it does not represent the full scope of FHE. Early systems like RSA demonstrated the potential for homomorphic properties, but they were limited, supporting only specific types of operations on ciphertexts.

The realization of FHE schemes, where an arbitrary function can be performed on encrypted data, remained elusive until Craig Gentry's groundbreaking work in 2009 [14]. Gentry's research marked a turning point, demonstrating for the first time how to construct a system that could handle an unlimited number of both addition and multiplication operations on encrypted data by introducing the concept of *bootstrapping*. Since then, the field has seen a proliferation of new constructions and schemes, each contributing to the advancement and practical feasibility of FHE.

The applicability of FHE extends across numerous sectors, enabling secure data processing within cloud environments by effectively preventing unauthorized data extraction through encryption. In the healthcare industry, for instance, FHE empowers researchers to securely analyze encrypted medical records, deriving critical insights without compromising patient confidentiality. Similarly, in the financial domain, FHE plays a pivotal role by allowing the secure processing of encrypted financial transactions and their analysis with machine learning in unregulated cloud-based environments.

However, a significant limitation of FHE is its substantial computational overhead. Consequently, this paper investigates possible use cases and the practical applicability of FHE.

Organization: The remainder of the paper is organized as follows: In section 2, Fully Homomorphic Encryption (FHE) is thoroughly defined, analyzed, and positioned within the field of cryptography. Specifically, subsection 2.1 discusses the properties of FHE, and subsection 2.2 defines various types of homomorphic encryption. Subsection 2.3 delves into the historical development of FHE, allowing subsection 2.4 to provide a brief overview of the methods for achieving FHE. Subsection 2.5 conducts a comprehensive security analysis of FHE, followed by an examination of its limitations in subsection 2.6. Homomorphic encryption is then situated within the cryptographic field and delineated from other methods in subsection 2.7. Subsection 3 offers a brief overview of current encryption schemes and their implementations. Subsequently, section 4 defines potential use cases, thereby concluding the theoretical analysis of FHE in general. Section 5 describes the new contributions in this paper and outlines the practical work following this survey.

Preliminaries: No specific prior knowledge is required to read this paper. A basic understanding of different security definitions is helpful but not essential.

Notation: In this paper, m generally denotes a plaintext message, and c represents the corresponding ciphertext. The symbol f is used to denote an arbitrary function/circuit from the function/circuit space C. The security parameter of the encryption schemes is denoted by λ, while the various keys in homomorphic encryption schemes are represented by SK, PK, and EK for the secret, public, and evaluation keys, respectively. For traditional encryption methods, k_{priv} and k_{pub} are used to denote the private and public keys. The homomorphic encryption procedures encryption, decryption, and evaluation are denoted by Enc, Dec, and Eval, respectively. For improved readability, the specification of individual parameters of these procedures are sometimes omitted. In contexts where both classical and homomorphic encryption are utilized, the subscripts serve to specify the encryption scheme in question. For example, Enc_{normal} and Enc_{FHE} are used to distinguish between non-homomorphic and homomorphic encryption, respectively. The symbol p denotes an arbitrary polynomial, h represents any hash function, d indicates the depth of a function/circuit, Pr denotes a probability, and "negl" stands for a negligible probability. At the level of individual operations, $+$, $\cdot$, and $*$ denote the plaintext operations of addition, multiplication, and an arbitrary plaintext operation, respectively. The equivalent operations on the corresponding

ciphertexts are denoted by $\oplus, \odot$, and $\star$. For an arbitrary function f we do not differentiate between the evaluation of f on plaintexts and ciphertexts.

2 HOMOMORPHIC ENCRYPTION

An encryption scheme[1] is called homomorphic over an operation if given some ciphertexts, the operation over the plaintexts can be performed without decryption by manipulating the ciphertexts directly [22]. Formally (correct) homomorphic encryption over an operation is defined as follows.

Definition 2.1 ((Correct) Homomorphic Encryption). Given an encryption scheme with the encryption function Enc, the decryption function Dec and two plaintexts m_1, m_2 with their respective ciphertexts $c_1 = \text{Enc}(m_1), c_2 = \text{Enc}(m_2)$. The encryption scheme is called *(correct) homomorphic over an operation* $*$, if the operation $*$ on the plaintexts can be directly performed with its analog operation $\star$ on the ciphertexts such that

$$m_1 * m_2 = \text{Dec}(\text{Enc}(m_1) \star \text{Enc}(m_2)) = \text{Dec}(c_1 \star c_2)$$

holds.

The ability to evaluate *any* function homomorphically is achievable if addition and multiplication can be performed homomorphically and can be iterated, since they constitute a functionally complete set over finite rings. In particular, any boolean (arithmetic) circuit can be expressed solely through the use of XOR (addition) and AND (multiplication) gates [1]. Bitwise addition and multiplication are thus regarded as foundational operations within FHE schemes.

Note: Rotation is also considered a base operation in FHE schemes. This operation, particularly when applied to a ciphertext containing multiple packed plaintexts, facilitates the rearrangement of various plaintext slots. Such a capability significantly contributes to the optimization of implementations for higher-level computational operations.

is defined as a tuple of four probabilistic algorithms, namely

Formally correct a Homomorphic Encryption (HE) scheme as a tuple of four probabilistic algorithms, namely KeyGen, Enc, Dec, and Eval. The KeyGen algorithm essentially takes as input the security parameter λ and outputs a secret key SK, a public key PK, and an evaluation key EK. The Enc procedure takes a message m, encrypts the message under the public key PK and outputs the ciphertext c. The Dec algorithm decrypts a ciphertext c to a plaintext m, given the secret key SK.

The Eval procedure is what makes HE schemes special. Given the evaluation key EK, a function f, and a ciphertext c the Eval procedure outputs a ciphertext c'. We assume a correct HE scheme here, so the output of the Eval function is corresponding to the functioned plaintexts, so the following holds:

$$\text{Dec}(c') = \text{Dec}[\text{Eval}(\text{EK}, f, c)] = f(m).$$

The ciphertext c and the plaintext m can be a vector of cipher- or plaintexts, e.g. $c = (c_1, c_2)$. FHE schemes also have the additional "Refresh" procedure that takes a ciphertext c_1, the evaluation key EK and a multi-valued flag as inputs and returns a new ciphertext c_2 that encrypts the same plaintext as c_1. The desired property of the Refresh function is to transform a complex ciphertext into a "simple" one, allowing more homomorphic operations to be performed on the fresh ciphertext. "Therefore either the bootstrapping procedure (flag = "Bootstrap") is performed, which takes a ciphertext with large random error (*noise*) and outputs a new ciphertext of the same message with a fixed amount of noise, or the key-swichting procedure (flag $\in$ {Relinearize, ModSwitch}) is applied, which takes a ciphertext under one key and outputs a ciphertext of the same message under a different key"[4]. In Table 1 a short overview of the keys and procedures of normal encryption compared to homomorphic encryption is given.

[1]This paper only covers asymmetric homomorphic encryption schemes. It does not discuss symmetric FHE schemes, given their relatively restricted utility in cloud computing.

Table 1: Comparison of HE vs. normal encryption
The Refresh **procedure is only needed for FHE schemes and is often avoided in LHE.**

		classic encryption	homomorphic encryption
keys	SK	•	•
	PK	•	•
	EK	○	•
procedure	KeyGen	•	•
	Enc	•	•
	Dec	•	•
	Eval	○	•
	Refresh	○	•

Note that this representation is greatly simplified. There are a few more procedures defined in the "Homomorphic Encryption Standard" [4] and the plaintext space, the ciphertext space and the output space of the Eval function can differ. Here we omit the difference between the ciphertext space and the output space of the Eval function for simplicity, and because the evaluation of the identity function can transform a ciphertext into the output of the Eval function. An extensive discussion about the formal definition of homomorphic encryption schemes and their properties is given in the paper of Armknecht et al. [6].

2.1 Attributes

For each FHE scheme the basic properties of *correctness, compactness* and *circuit privacy* should hold [6]. To have a closer look at this properties we first have to define the term C-homomorphic encryption scheme.

Definition 2.2 (C-Homomorphic Encryption Scheme). A HE scheme is called C-homomorphic encryption scheme if it is an *efficient, secure* homomorphic encryption scheme over each function in C.

Secure means in this context IND-CPA secure (see Definition 2.9). *Efficiency* guarantees that the KeyGen, Enc and Dec procedure of the scheme run in polynomial time in relation to the security parameter λ (and is not dependant on the function that is evaluated on the ciphertext) [4]. It is important to note, however, that this is a theoretical requirement and, in practical terms, may permit runtimes that are exceedingly impractical.

Table 2: Circuit Privacy vs. Function Privacy

Privacy	Distributions of ... are the same	
Circuit	Eval output of f_1	fresh ciphertexts
Function	Eval output of f_1	Eval output of f_2

The functions in C are also called *circuits*[2] and some authors also use the term C-evaluation scheme instead of C-homomorphic encryption scheme.

Definition 2.3 (Correctness). A C-homomorphic encryption scheme is *correct*, if it can decrypt the encryption of a message without any error and if for all functions $f \in C$, it can correctly decrypt the results of the evaluation of f over fresh ciphertexts with overwhelming probability [3] [6].

Note: For intuitive purposes, the definitions of Homomorphic Encryption and, by extension, a C-Homomorphic Encryption scheme, presume their correctness. Formally, a Homomorphic Encryption scheme and a C-Homomorphic Encryption scheme may not necessarily fulfill this criterion of correctness and are just defined as a tuple of probabilistic polynomial–time algorithms
(KeyGen, Enc, Eval, Dec), where the Eval procedure just returns a ciphertext c'. We posit that a homomorphic encryption scheme, if deemed incorrect, cannot be considered homomorphic by its very nature. Thus, we have intuitively assumed its correctness in our definitions. See Figure 1 for a formal correct overview. Also note that we always assume fresh ciphertexts here. See subsection 2.2 for further explanations to the i-hop properties of homomorphic encryption schemes.

Definition 2.4 (Compactness). Given the security parameter λ, a C-homomorphic encryption scheme is *compact* if there is a polynomial p, such that for all possible keys, all $f \in C$ and all possible ciphertexts the size of the output of the Eval function is not bigger than $p(\lambda)$ bits, independent of the complexity of the evaluated function f [6].

That means ciphertext growth is only dependant on the security parameter.

Definition 2.5 (Circuit Privacy). A C-homomorphic encryption scheme is (perfectly, statistically or computationally) *circuit private* if for any keys, any function $f \in C$, any fresh ciphertexts c with $\text{Enc}(m) = c$ the distribution of the evaluation of f over the ciphertexts is the same as the distribution of the encryption of the evaluated plaintexts under the function f.[4] In formulas this implies that the distributions $D_1 = \text{Eval}(\text{EK}, f, c)$ and $D_2 = \text{Enc}(\text{PK}, f(m))$ should be (perfectly, statistically or computationally) indistinguishable [6].

Because $f(m)$ is just another plaintext, circuit privacy implies that an attacker can not distinguish between a fresh ciphertext and the output of the evaluation procedure. *Function privacy* is a weaker requirement than circuit privacy, where only the indistinguishability between the output distributions of different evaluation functions on ciphertexts is demanded. Function privacy is sometimes also called *evaluation privacy* and implies that no information about f beyond the outputs for the queried inputs is revealed. An overview of the comparison between circuit and function privacy is given in Table 2. "Note that for a scheme to be circuit or function private, the property has to hold even against an adversary that knows the secret key and can decrypt any ciphertext." [22]

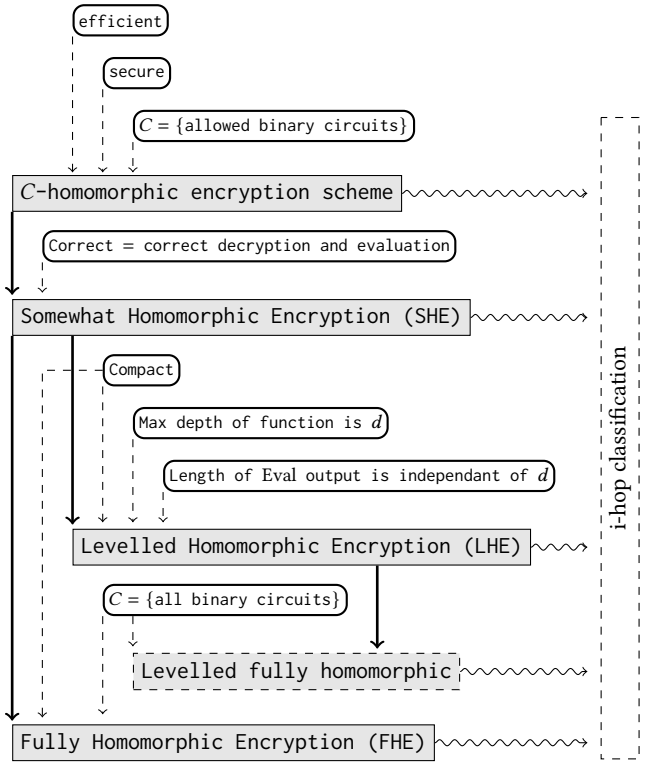

Figure 1: Properties are rectangles with rounded corners, classes are rectangles with gray background color. A dotted arrow indicates, that the property holds for a class and a thick arrow indicates that a class evolves from another. The i-hop classification of the schemes is orthogonal to the other classification. The figure is inspired by Armknecht et al. [6].

2.2 Classification

Now after the basic properties of homomorphic encryption schemes have been discussed, the different types of homomorphic encryption schemes can be defined. In general, HE schemes can be structurally divided into various types and historically categorized in different generations.

In *practice* (informally) there are three structurally different types of HE, namely partially, somewhat/levelled and fully homomorphic encryption. Partially Homomorphic Encryption (PHE) is limited to just one type of operation, either addition or multiplication. Most of the known PHE schemes support any number of operations (either

[2] Often the term circuit is used instead of function because the addition and multiplication are performed bitwise, meaning addition is equivalent to XOR and multiplication is equivalent to AND. This means every function is represented by a boolean circuit.
[3] testtest
[4] Circuit private is sometimes also called "strongly homomorphic".

addition or multiplication). "Somewhat Homomorphic Encryption (SHE) supports mathematical operations with respect to adition and multiplication, but is limited to a certain number of operations (since each operation adds noise and after a certain amount of noise is added, it is no longer possible to retrieve the data)." [25] Finally, FHE supports both addition and multiplication, applied any number of times to the data, therefore allowing the evaluation of any function.

Since homomorphic encryption has not yet been standardized and the distinction between SHE and LHE is more theoretical in nature, they are sometimes used synonymously. In a formal correct way, however, there is a difference between these two types of schemes and the additional distinction between the set of homomorphically evaluable functions/ circuits C and the ability to perform a homomorphic operation on the output of the Eval function have to be made. For extensive formal definitions of the various types of homomorphic encryption, please refer to the paper by Armknecht et al. [6].

Here only a brief overview of the formally correct definitions is given. Given the formal definition, SHE and LHE schemes are not constrained to support addition and multiplication but just an arbitrary set of functions C. This means they can also be partially homomorphic in the informal sense.

Definition 2.6 (Somewhat Homomorphic Encryption (SHE)). A *SHE* scheme is defined as a correct C-homomorphic encryption scheme that does not necessarily have to be compact.

Definition 2.7 (Levelled Homomorphic Encryption (LHE)). A *LHE* scheme is a correct, compact C-homomorphic encryption scheme that allows only functions of a certain depth [5] d given by an auxiliary input. Additionally, the length of the evaluation output must not depend on d.[6]

The difference between those two types of schemes is that SHE schemes do not have to be compact, so evaluating functions of a higher depth can also increase the output length of the evaluation function. LHE schemes on the other hand are compact and the depth of functions that can be evaluated is a parameter on which the length of the evaluation output does not depend.

Definition 2.8 (Fully Homomorphic Encryption (FHE)). A *FHE* scheme is defined as a compact, correct C-homomorphic encryption scheme where C is the set of all circuits.

This means FHE are turing complete, allowing to evaluate any function on ciphertexts homomorphically.

The informal definitions could suggest that HE schemes allow to perform a homomorphic operation on the output of the Eval function. In the formal definition however the schemes have to be correct, so the evaluation on *fresh* ciphertexts is guaranteed to work but not on the output of a different evaluation. To show that a HE scheme allows for the evaluation of functions on the output of the Eval function, the concept of *i-hop correctness* is needed. Informal i-hop correctness essentially means that a HE scheme is capable of processing the output of the Eval function i times.

Note: Evaluating an arbitrary function is not equal to consecutively evaluating arbitrary many functions. Consider

$$f(\dots(f(m))) \coloneqq F_n(m).$$

$F_n(m)$ is an arbitrary function on the plaintext m, so FHE can perform Eval(EK, F_n). But FHE does not necessarily allow to perform Eval (EK, f (... (Eval(EK, f)))) because Eval(EK, f) could map the plaintext m to a different message space, so the application of Eval(EK, f) again is not valid anymore. The property of i-hop correctness is particularly important in scenarios where different entities work together on processed data of someone else without having a fresh encryption. In practice, FHE schemes are always 1-hop correct after the bootstrapping procedure.

An overview of the formal definitions of HE schemes is given in Figure 1. For further information refer to Armknecht et al. [6].

Note: Numerous PHE schemes are known today, including RSA (1978, multiplicative), El-Gamal (1985, multiplicative), Goldwasser and Micali (1982, additive), and Benaloh (1994, additive) [1]. These schemes offer faster computation compared to FHE schemes. However, their support for only one type of operation limits their application to basic statistical calculations like counting, mean, or standard deviation in a secure manner. The aforementioned schemes rely on the factorization problem, the discrete logarithm problem, or a residue problem. This reliance makes them vulnerable or potentially vulnerable to attacks by quantum computers. Consequently, this paper will not explore PHE schemes further. For additional information on PHE schemes, please refer to the paper by ACAR et al. [1] and the book by Koç et al. [19]. An overview of existing PHE schemes, their supported operation, and additional information is given in the appendix in Table 11.

2.3 History

Here we only give a short overview of the history of FHE that can be divided into four generations [22]. In general, all known FHE schemes today add some noise during the encryption process that increases with each homomorphic operation until a certain threshold is reached and the ciphertext is not decryptable any more. To reduce the noise growth and the absolute noise of an evaluation output different techniques have been proposed. The FHE generations differ initially in their underlying mathematical problems and later in the techniques used to limit noise growth and refresh ciphertexts. A timeline showing the most important schemes of each generation is shown in Figure 2.

The first FHE scheme was proposed by Craig Gentry in 2009 [14]. It is based on ideal lattices and introduced the *bootstrapping* procedure to evaluate functions of any depth on ciphertexts. Although the scheme supports batching. It is very slow in practice due to the fast noise growth. Additionally, it is hard to implement and vulnerable against key recovery attacks. The DGHV scheme is another scheme of the first generation. It was proposed by Dijk et

[5] The depth of a function or circuit is defined as the maximum number operations applied to the input. In practice, the depth is often just the number of multiplications and/ or additions applied to the input.

[6] "If we require that C is the set of all binary circuits with depth at most d, the scheme is called levelled fully homomorphic." [6]

[7] Note: This is a greatly simplified historical overview. Some authors may use slightly different dates, as they consider other papers to be the introduction of the encryption schemes, or because they make a finer distinction between different schemes.

[8] CKKS has a fast amortized bootstrapping procedure.

Table 3: Comparison of FHE generations

SCHEMES		2nd Generation BGV	2nd Generation BFV	3rd Generation TFHE	4th Generation CKKS
		Integer Arithmetic		Bitwise operations	Real Number Arithmetic
FAST OPERATIONS	scalar multiplication	●		●	●
	arithmetic	●		●	●
	non-arithmetic	○		●	○
PROPERTIES	fast bootstrapping	○		●	● [8]
	fast packing/ batching/ SIMD	●		○	●
	levelled design	●		●	●
PROS	fast	scalar multiplication linear functions		number comparison -	polynomial approx. multiplicative inverse
	efficient	-		boolean circuits	DFT, logistic regression
CONS		slow non-linear functions		-	slow non-linear functions
USAGE		large arrays of numbers		bitwise operations	real numbers arithmetic

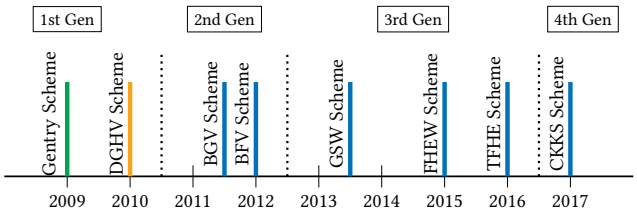

Figure 2: Timeline of the main FHE schemes.
■ Schemes based on ideal lattices, ■ Schemes based on AGCD, ■ Schemes based on LWE and RLWE [7]

al. [26] in 2010 and is based on the Approximate - Greatest Common Divisor (AGCD) problem. It suffers from big public keys and high computational complexity. To reduce the public key sizes the *modulo switching* technique was introduced. Both schemes of the first generation are not relevant today because their noise growth negatively affects efficiency and security.

Nearly all the schemes following the first generation are based on Learining with Error (LWE) or Ring Learining with Error (RLWE) (with some exceptions based on NTRU that are not discussed here) which leads to better-understood security assumptions. The most important schemes of the second generation are BGV (2012) [8] and BFV (2012) [13]. "The Brakerski-Gentry-Vaikuntanathan (BGV) and Brakerski/ Fan-Vercauteren (BFV) schemes are the two main HE schemes to perform exact computations over finite fields and integers." [18] With this generation, the techniques *relinearization* and *modulus switching* were introduced. BGV and BFV allow better noise control, higher efficiency, a better plaintext to ciphertext ratio (named *packing*, which allows single instruction multiple data (SIMD) instructions) and optimizations on the bootstrapping procedure [16]. *Scale invariant* is one technique of this generation, to reduce the noise growth in a variant of the BGV scheme from exponential to linear, which eliminates the need for modulus switching. NTRU-based encryption schemes are not discussed in this paper primarily due to the necessity for significantly increased parameters for securing such schemes against recent attacks. This increase in parameters has led to NTRU-based schemes becoming much less efficient compared to their counterparts, resulting in their diminished use and lack of support by any existing library [22].

The third generation started in 2013 with the introduction of the GSW scheme and includes the GSW (2013) [15], the FHEW (2015) [12] and TFHE (2016) [10] scheme. These schemes have a different noise growth pattern compared to the schemes of the second generation and are, according to Shai Halevi, less efficient but therefore need weaker hardness assumptions [16]. With GSW the *approximate eigenvector method* was introduced, which eliminates the need for key and modulus switching techniques by reducing the error growth of homomorphic multiplications. With TFHE, the bootstrapping procedure and a function evaluation became possible in one step which is known as *programmable bootstrapping*.

The fourth generation of FHE starts in 2017 with the CKKS encryption scheme [9]. In general, CKKS is similar to the schemes of the second generation but it uses approximate computation, which is considerably faster and it allows floating-point arithmetic (by homomorphically operating over approximations of real numbers), which is necessary for most machine learning algorithms.

In general, schemes of the first generation are not relevant today. Schemes of the second generation are good at performing exact computations over integers on large arrays of numbers simultaneously because the schemes allow SIMD operations, but they are not good for depth functions where bootstrapping is required. Third generation schemes, namely TFHE, can outperform previous schemes for bitwise operations but it does not support batching, hence the scheme can be outperformed when large amounts of data should be processed simultaneously. This allows CKKS to have a faster amortized bootstrapping procedure than TFHE, although TFHE has the fastest bootstrapping procedure. In general, "the fourth generation, i.e. CKKS, is the best option for real numbers arithmetic." [22] A final comparison of the FHE generations can be found in Table 3.

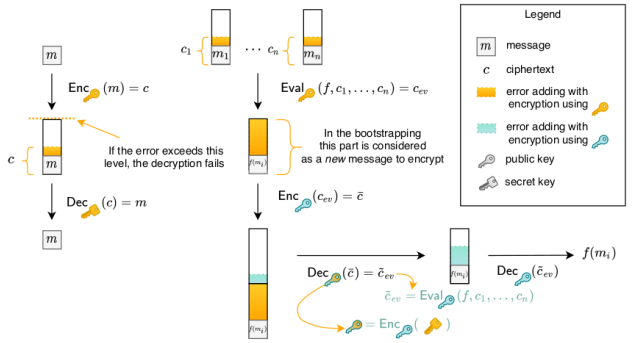

Figure 3: Illustration of the *bootstrapping* technique by Marcolla et al. [6]
In the left column, a normal encryption and decryption is shown. In the right column, the bootstrapping recryption procedure is illustrated.

Through the Chimera framework [7], switching between different encryption schemes such as TFHE, BFV, and CKKS is possible, allowing the benefits of all schemes to be leveraged. For an extensive overview of the most important schemes, their optimizations and relations, please refer to the paper by Marcolla et al. [22].

2.4 From SHE to FHE

Generally, all contemporary FHE schemes introduce a certain degree of noise during the encryption process. This noise accumulates progressively with each homomorphic operation, eventually reaching a critical threshold beyond which the ciphertext becomes undecipherable. This inherent characteristic constrains the depth of functions that can be evaluated. To surmount this limitation and attain a true FHE scheme, a refresh procedure is required to update the evaluation output, simplifying it for continued operations. According to the "Homomorphic Encryption Standard" [4] the Refresh procedure follows one of three known techniques namely "bootstrapping", "re-linearization" or "modulus switching".

Bootstrapping is a "recryption which works by encrypting a ciphertext anew (so that it becomes doubly encrypted) and then removing the inner encryption by homomorphically evaluating the doubly encrypted plaintext and the encrypted decryption key using the decryption circuit." [6][9] This technique only works, if the evaluation algorithm can perform the decryption plus one functional complete gate (e.g. NAND). The basic idea of bootstrapping is shown in Figure 3. The bootstrapping procedure takes a ciphertext with a large noise and outputs a new ciphertext of the same message with a smaller fixed amount of noise.

Note: Bootstrapping creates a large computational overhead because every ciphertext has to be recrypted.

In Gentry's first scheme an additional technique named *squashing* was introduced to reduce the complexity of the decryption circuit before bootstrapping. This technique was not adopted by later schemes and is therefore not discussed here.

[9]The encryption of the decryption key can be done by its own public key or another key. See subsection 2.5 for further information.

Depending on the scheme, the output size of the evaluation is bigger than the size of fresh ciphertexts. With *re-linearization*, also called *key-switching*, the ciphertext size is reduced back to normal. *Dimension-modulus reduction*, also called *modulus switching* is a technique to convert a ciphertext c mod q to c' mod p where p is sufficiently smaller than q.

In general, these techniques allow the transformation from a SHE scheme to a FHE scheme by updating the evaluation output. This theoretically allows a FHE scheme to evaluate any function on ciphertexts. In practice, it is beneficial to sidestep these resource-intensive techniques by limiting the depths of the functions to be evaluated on ciphertexts.

2.5 Security

This section provides a general overview of security notions in the context of FHE. No encryption scheme-specific security analysis is performed. This means that no overview of the underlying problems of LWE/ RLWE, their attacks, or recommended parameters is given. For further information on lattice based problems refer to the paper of Marcolla et al. [22]. More information on attacks on lattice-based problems, their runtime, and the resulting parameter recommendations can be found in the "Homomorphic Encryption Standard" [4] and the Estimator tool of Albrecht et al. [5].

A HE scheme is secure, if it is *semantically secure* (IND-CPA secure). Indistinguishability under chosen plaintext attack (IND-CPA) secure means that although an attacker is allowed to ask an oracle a polynomial number of times for the encryption of arbitrary plaintexts (but he can not ask for the decryption of a ciphertext), he can not distinguish between the encryption of 0 and 1 with non-negligible probability.[10]

IND-CPA security is formally defined as follows [22].

Definition 2.9 (IND-CPA Security). Let $\mathcal{E} = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ be a HE scheme, and m_b a message with $\{0, 1\}$ as the message space. Let us define an adversary $\mathcal{A}$ that knows the evaluation key EK and the public key PK and is given an encryption $\text{Enc}_{\text{PK}}(m)$ for $m \in \{0, 1\}$. $\mathcal{A}$ can make queries to the encryption oracle. After a polynomial number of queries, $\mathcal{A}$ tries to guess whether $m = 0$ or $m = 1$. Then, the scheme is *IND-CPA* secure if for an efficient adversary $\mathcal{A}$, it holds that:

$$\Pr\left[\mathcal{A}\left(\text{PK}, \text{EK}, \text{Enc}_{\text{PK}}(0)\right) = 1\right] - \Pr\left[\mathcal{A}\left(\text{PK}, \text{EK}, \text{Enc}_{\text{PK}}(1)\right) = 1\right] = \text{negl}(\lambda)$$

where $(\text{SK}, \text{PK}, \text{EK}) \leftarrow \text{KeyGen}(\lambda)$.

Theorem 2.10. *IND-CPA security is only achievable if the encryption scheme randomizes ciphertexts.*

Proof. If there is no randomization, an attacker could simply ask for the encryption of a message and compare the encrypted output with the given ciphertext. In the definition above the message space was restricted to $\{0, 1\}$, so the attacker just has to ask the orcale for the encryption of 0 and 1 and can then compare the ciphertexts to determine which message was encrypted. [11] □

[10]Note: The Enc − and Dec − procedure, the public key PK and the Eval key EK are public. Only the secret key SK is secret.
[11]Some PHE schemes are not randomized, so they should not be used for homomorphic encryption (e.g. RSA).

According to Acar et al. [1] some SHE schemes have been proven to be indistinguishability under (non-adaptive) chosen ciphertext attack (IND-CCA1) secure, but no unbounded FHE scheme is yet proven to offer this type of security.

Theorem 2.11. *By their design, HE schemes can not achieve indistinguishability under adaptive chosen ciphertext attack (IND-CCA2) security.*

Proof. IND-CCA2 security means, given a ciphertext c, an attacker is allowed to ask an oracle for the decryption of any ciphertext $c' \neq c$.

Given the ciphertext $c = Enc(m)$ the attacker encrypts a message m' ($\text{Enc}(m') := c'$) and adds it to c, resulting in $\overline{c} = c \oplus c'$. Now the attacker asks the oracle for the decryption of of $\overline{c} \neq c$. Since the encryption scheme is homomorphic, the following applies:

$$\text{Dec}(\overline{c}) = \text{Dec}(c \oplus c') = \text{Dec}(c) + \text{Dec}(c') = \text{Dec}(c) + m'.$$

Because m' and $\text{Dec}(\overline{c})$ are known, the attacker can determine the value of the original message m, which means the encryption scheme can not be IND-CCA2 secure. □

The lack of IND-CCA2 security has to be considered in the design of FHE protocols, especially when they handle some ciphertext in a special way. Consider the following example.

Example 2.12. An encryption scheme only allows the decryption of messages > 0. Otherwise, the decryption algorithm returns "FAIL". While watching the behavior of the decryption algorithm a malicious server sends the false evaluation result

$$c \oplus \text{Enc}(-t),\ t = 1, \ldots, n,$$

until the decryption fails the first time. [12] Since the encryption scheme is homomorphic and the decryption fails when 0 was encrypted, it follows:

$$0 = \text{Dec}(c \oplus \text{Enc}(-t)) = \text{Dec}(c) + \text{Dec}(\text{Enc}(-t)) = \text{Dec}(c) - t.$$

The attacker then knows that the original message was t.

The example demonstrates that when applying homomorphic encryption methods, it is crucial that an attacker cannot decrypt a certain class of ciphertexts (for instance by observing the behavior of the client during decryption) since they could be adaptively chosen by homomorphically changing given ciphertexts. This problem is nonexistent for non-homomorphic encryption schemes because in general

$$\text{Dec}(c) + \text{Dec}(\text{Enc}(m)) \neq \text{Dec}(c \oplus \text{Enc}(t)) = \text{nonsense}$$

holds, which allows no information retrieval. Countering such attacks is beyond HE and can be solved on the security protocol level (c.f. sloppy Alice attack on the McEliece encryption scheme and its counterattacks). A robust protocol could for example transmit the same data in a newly encrypted form instead of answering "FAIL" after the decryption of a function evaluation result on the send data failed. Due to the variance in the added noise with each encryption, the attacker remains unaware that the newly transmitted data is identical to the previously sent data. This method effectively mitigates the demonstrated attack, rendering it non-threatening in this context.

Note: Knowing arbitrary pairs $(m, c = \text{Enc}(m))$ does not allow for the demonstrated type of attack because FHE schemes offer IND-CPA security. For the demonstrated attack an attacker has to be able to decrypt an adaptively chosen ciphertext. Comparing the encryption $\text{Enc}(m + m')$ to a given ciphertext $\overline{c}$ or comparing $\overline{c}$ with $c \oplus \text{Enc}(m')$ for different m' is not feasible because different random noise is added.

The previous considerations focused on HE in general. For FHE an additional security assumption, namely *Circular Security* is needed.

Definition 2.13 (Circular Security [22]). An encryption scheme that is secure against adversaries who observe an encryption of the scheme's secret key under its public key is called *circular secure*.

All today-known FHE constructions rely on bootstrapping, which is only possible if an encryption of the secret key under its own public key is published. "This implies that IND-CPA security has to hold under circular security. Most FHE schemes are not proven IND-CPA secure under circular security, and it is in general adopted as an additional assumption on top of the scheme's underlying security assumptions." [22]

If the circular security assumption does not hold, it is also possible to use a chain of private/ public keys $(\text{SK}_i, \text{PK}_i),\ i = 1, \ldots, n$ and publish only the encryption of a secret key under the next public key. This means $\text{Enc}(\text{SK}_i, \text{PK}_{i+1})$ is published instead of $\text{Enc}(\text{SK}_1, \text{PK}_1)$. *Note*, that the resulting scheme is limited in depth by the number of key pairs. This is in practice no constraint [13] and eliminates the circular security assumption. However a weaker security assumption, called *KDM security*, is needed.

Definition 2.14 (Key Dependent Message (KDM) [21]). An encryption scheme is Key Dependent Message (KDM) secure if it is secure even against an attacker who has access to encryptions of messages which depend on the secret key.

The difference between KDM and circular security in the context of bootstrapping is that the first only publishes an encryption of the secret key while the latter publishes an encryption of the secret key under its own public key. Thus, the KDM security assumption contains less potential structure for attacking the encryption of the secret key.

For further information on KDM security and its proof, we refer to the paper by Malkin et al. [21]. To eliminate also this assumption in some LHE schemes modulus switching can be used to reduce the noise. For optimizations bootstrapping is still often recommended, adding the KDM security assumption again.

The security concepts discussed before guarantee the confidentiality of the data but they do not prevent an attacker from delivering wrong evaluation results. Consider a client who wants to securely outsource the computation of a function f on encrypted data c to the cloud. Instead of $\text{Eval}(f, c)$ the malicious cloud provider

[12] If the decryption fails the first time $t = -1, \ldots, -n$ has to be tested until the decryption does not fail any more.

[13] If the Evaluation is performed by a server and the max depth is reached, this could be communicated to the client, who then sends new key pairs. This eliminates the depth constraint in practice.

returns Eval(g, c) for some function $g \neq f$. This means an active attacker could return false evaluation results of potentially significant consequences. The intention behind such an attack might not necessarily be to harm the client, but could be motivated by the cloud provider's desire to conserve their own resources by evaluating a simpler function g instead of f.

To overcome this threat multiple techniques can be used.

1. The easiest way would be to include data with known evaluation results. This means that the client has to perform the computation on its side and compare it to the results by the cloud provider. The higher the percentage of pre-computed evaluations that are compared, the higher the security.

2. Another way is to run statistical tests on the returned outputs and look for significant changes that could trigger the need for the first technique. This could be additionally supported by a live benchmark of the server to ensure a function of similar complexity as the given one was evaluated. This would eliminate the incentive to save resources.

3. Hardware techniques like Trusted Execution Environments could also be used to ensure the right function was evaluated on the data. This technique however needs specialized hardware that is likely to increase the cost of outsourcing FHE computations. In research, the focus therefore lies on mathematical-based solutions that can run on every hardware.

Another way could be the use of homomorphic hashes or signatures. The idea is that the client publishes the ciphertext c, the function f that should be evaluated and a homomorphic hash function h. The cloud evaluates the function on the encrypted data and returns $c_{res} = f(c)$. The client can then check if the evaluation was performed as expected by calculating $h(c_{res}) = h(f(c))$ and check if it matches f(h(c)). This technique is relatively new and adds an additional computational overhead, so it is not discussed in this paper.

Note that the discussed threat of unwanted transformations of ciphertexts (to other valid ciphertext) is also known as the *malleability property*.[14] It is also present during the transportation of data to a third party (e.g. cloud provider). An attacker could for example alter the content of "Send 100$ to bank account 1234" to "Send 100$ to bank account 5678" by homomorphic addition. To mitigate this attack during transportation, the FHE encrypted data c must be send with an additional normal signature

$$sig(c) = \mathrm{Enc}_{\mathrm{normal}}\left(h(c), \mathrm{k}_{\mathrm{priv}}\right).$$

In this context, h denotes a normal (non-homomorphic) hash function and $\mathrm{k}_{\mathrm{priv}}$ a normal private key of the client (not the already used one private key SK of the homomorphic encryption). The server can check the authenticity and integrity of the data by checking the signature. The final send message is then

$$(c, sig(c)) = (\mathrm{Enc}_{\mathrm{FHE}}(m, \mathrm{PK}), sig(c)).$$

Instead of just sending the FHE encrypted message with its signature, the homomorphic ciphertext can be sent by using a normal encrypted transportation protocol that offers authenticity and integrity of the data. The advantage of this technique is that an existing secure and standardized solutions can be used. A small drawback is that the send message is slightly bigger because another layer of encryption is added. Let $\mathrm{k}_{\mathrm{pub}}$ be the normal public key of the server and PK the FHE public key of the client. In formulaic terms, an additional encapsulation with standard transport encryption means that

$$\begin{aligned}&\left(\mathrm{Enc}_{\mathrm{normal}}(c, \mathrm{k}_{\mathrm{pub}}), sig(c)\right)\\ =&\left(\mathrm{Enc}_{\mathrm{normal}}(\mathrm{Enc}_{\mathrm{FHE}}(m, \mathrm{PK}), \mathrm{k}_{\mathrm{pub}}), sig(c)\right)\end{aligned}$$

is transmitted.

[14] Non homomorphic encryption schemes are often non malleable. (See explanation below Theorem 2.12)

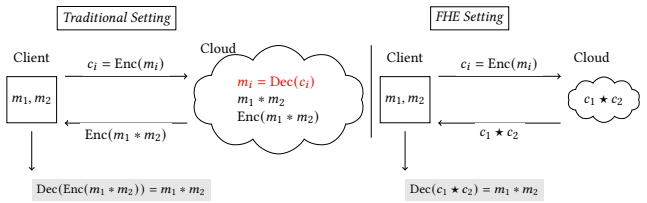

Figure 4: Usage of FHE in the public cloud: In the traditional setting on the left the ciphertexts have to be decrypted for computations in the cloud, thereby posing a risk of data breaches. This risk is not present in the FHE setting on the right because calculations can be directly performed on encrypted data without the need of decryption.
In the graphic, m_i represents the plaintext, Enc **and** Dec **denote the encryption and decryption function, $*$ stands for any operation on plaintexts and $\star$ for the according operation on ciphertexts.**

In our previous analysis, we observed that HE entails negative implications for security due to its inherent homomorphy. However, it is noteworthy that HE enhances security in outsourced computations. This enhancement stems from the capability of FHE to facilitate encryption during processing, extending beyond the traditional confines of encryption at rest and in transit, as offered by conventional encryption methodologies. Consequently, this enables the secure offloading of computations to the cloud, obviating the need to decrypt the data intermittently. Figure 4 shows the advantage of FHE over traditional encryption.

An additional feature of modern FHE schemes like BGV [8], BFV [13], FHEW [12], TFHE [10] and CKKS [9] is that they are considered to be quantum-safe because they are based on LWE or RLWE. For the entire encryption scheme to be quantum-safe, quantum security must also apply to the circular security assumption. But since the scheme is assumed semantically secure (also in the context of quantum computers), an adversary can not distinguish the encryption of the secret key from the encryption of an arbitrary plaintext, so the circular security assumption should also be quantum safe.

Additionally HE schemes can offer circuit or function privacy (Definition 2.5). According to Shai Halevi [16] many FHE schemes can also offer great *leakage resilience*[15] due to their underlying problems, namely LWE and RLWE.

An additional security feature of all sufficient HE schemes is that they allow *key evolution* [4].

[15] Leakage resilience means resilience against side-channel attacks.

Definition 2.15 (Key Evolution). Key evolution allows a client to replace a compromised secret key SK by a new secret key SK′, so that semantic security still holds even for an adversary who holds SK. The transformed ciphertexts can be decrypted only by the client using SK′.

The idea behind key evolution is the same as in bootstrapping. The difference is only that the old secret key SK has to be encrypted under a new public key PK′ and not its own. With key evolution, compromised keys can be exchanged without decrypting the data.[16] Key evolution thus facilitates the long-term storage of sensitive data in the cloud under regularly renewed keys. This approach further minimizes the risks associated with brute force attacks on the keys. For further information we recommend the "Homomorphic Encryption Standard" [4].

In general, there is always a tradeoff between security and runtime. In FHE, this tradeoff requires even more careful consideration, as poor parameter selection can either compromise the encryption's security or necessitate the extremely computation-intensive process of bootstrapping. Practically, this means that in contrast to conventional encryption methods, the security parameter should not be chosen liberally but as small as is securely feasible. A very useful and well-established resource for security-parameter selection is provided by the lattice-estimator tool, as referenced in [5].

The security analysis conducted was not specific to any particular encryption scheme and is applicable to HE in general. However, since CKKS is the method of choice in the use case specified later, we would like to make a note here regarding a procedure-specific impact on security. Li and Micciancio [20] revealed a vulnerability in the CKKS encryption scheme and its variants, where an adversary with encryption access could potentially extract the secret key through linear algebra or lattice reduction methods, especially if they have knowledge of both the decrypted elements and corresponding ciphertexts. To counteract this, they recommend against sharing the results of decrypted messages. However, for scenarios like secure multi-party computation where sharing plaintexts is essential, they propose a workaround: introducing an error at the decryption's result to prevent such attacks.

2.6 Limitations of FHE

The principle underlying Fully Homomorphic Encryption (FHE) renders it an ideal cryptographic solution for outsourcing computations to the cloud while maintaining data confidentiality. This raises the question as to why the technology has not yet become widespread in practical applications. There are three main reasons for this: firstly, the significant computational and storage overhead [6]; secondly, the lack of standardization; and lastly, the still complicated way to use FHE implementations [22].

1. Today's homomorphic encryption schemes are based on probabilistic algorithms to guarantee IND-CPA security by introducing an error termed *noise* into the encoded plaintext. With each homomorphic operation, the noise growths, thereby restricting the feasible number of operations on ciphertexts before decryption fails. To facilitate an indefinite number of operations on ciphertexts, resource-intensive techniques such as bootstrapping, squashing, modulus switching, or relinearization are essential. However, the inherent increase in noise is the basis of the security of these schemes but it also poses significant challenges in developing practical FHE schemes. Those resource-intensive techniques can be avoided by only evaluating low-depth functions on the encrypted data. Nevertheless, FHE remains highly computation-intensive and requires very large keys. A general comparison with plaintext computations is not presented here, as the runtimes can vary significantly depending on the scenario. The resource consumption (both in terms of runtime and memory) of FHE is heavily dependent on several factors: the specific HE method employed, the chosen security parameter, the particular implementation, the function being evaluated, and the encoding technique of the plaintext. In the master's thesis building upon this seminar paper, a comparison is conducted within a specific scenario, as detailed in section 5. In the future, as has been the case in the past, significant improvements in runtime and storage are expected. This will be achieved on two fronts: firstly, at the hardware level through the utilization of specialized hardware such as FPGAs, and secondly, at the software level through the implementation of more efficient packing techniques. Table 4 shows the historical improvements in the runtime of FHE multiplication to give a hint on possible future improvements.

Table 4: Running times of multiplying 2 bits homomorphically and the expected running time in 2024 according to Duality [11]

Year	runtime	speedup	speedup per year
2009	30 min	-	-
2014	2000 ns	$9 \cdot 10^8$	$18 \cdot 10^7$
2020	100 ns	20	3.33
	…Hardware Acceleration …		
2024	*0.1 ns*	*1000*	*250*

Another limitation of FHE is that its output is always encrypted and can only be decrypted with knowledge of the secret key.[17] This characteristic implies that the result of branching conditions is also encrypted, thereby making it impossible to process with unencrypted processors. Consequently, the data-independent execution inherent in FHE extends to branching (as well as to the termination conditions of loops), necessitating the evaluation of all branches. The correct branch can only be selected at the end of this process. A potential solution involves decrypting branching conditions on the client side [25]. However, this requires constant communication with the client and can significantly compromise security [25].

Another limitation of FHE, already mentioned in subsection 2.5, is that it does not guarantee integrity, meaning the client can not be sure that the right function has been evaluated on the encrypted data on the server side [16]. Additionally, it has to be mentioned that HE does not allow to keep the evaluated functions secret. As mentioned in Theorem 2.5 HE only offers that no information about the evaluated function can be retrieved from the output of the Eval procedure. But if a client sends for example an encrypted

[16] It is important to guarantee the authenticity of the data owner to prevent the adversary from exchanging the keys.

[17] This is in contrast to obfuscation and functional encryption. See subsection 2.7 for further information.

neuronal network and encrypted data to the cloud, only the weights of the network are encrypted. Its structure is fully disclosed to the server. Hiding the algorithm applied to the data is subject to obfuscation research [6]. See subsection 2.7 for further information. If the algorithm and the data are owned by different entities both the data and the algorithm can be kept secret. The hiding of the algorithm however results from the fact that the algorithm never leaves the secure environment of the algorithm owner (see Use Case 2). So secret function evaluation is not achieved by HE.

Some authors cite the requirement to encrypt all data under a single key as a limitation [16]. However, as this is also a characteristic of traditional encryption methods, we contend that this does not constitute a significant restriction in our view. Additionally, there are already methodologies supporting what is known as multi-key FHE (see Use Case 6).

2. First standardization efforts were initiated in 2017 by the consortium "HomomorphicEncryption.org," comprising industry, government, and academic researchers from entities such as IBM, Microsoft, Intel, the NIST, and others [4]. While the standardization process has not reached completion, preparatory documents have been issued by the ISO. These endeavors are anticipated to culminate in an established standard in the foreseeable future. However, as of the present, the absence of a finalized standardization remains a constraining factor, also limiting the trust in the technique. In the future, the standardization of FHE could be its main advantage, as FHE operates independently of data and can evaluate any function. This implies that any evaluation of private functions becomes standardized once FHE is standardized. In combination with stable open source libraries like OpenFHE this will likely increase the trust and the adoption of FHE. While specific use cases may have, and will continue to have much better privacy-preserving solutions, these solutions must be standardized within their respective data- or function-dependent scenarios.

3. To develop FHE applications at the moment, an extensive understanding of HE is essential [22]. This encompasses various aspects, including parameter selection, plaintext encoding, efficient packing, and approximating high-level functions with the basic FHE operations of addition and multiplication.[18] This highly limits the practical usability of FHE at the moment. Current developments in HE compilers and other high-level tools aim to automate the transformation of high-level programs, such as machine learning models, into FHE-based implementations. Consequently, FHE is becoming increasingly user-friendly, potentially enabling the fully automated conversion of existing algorithms into their FHE equivalents, thereby minimizing (or entirely eliminating) the need for additional engineering.

Beyond the additional engineering efforts required for developing FHE-compatible algorithms, key management systems must also be adapted, as FHE solutions necessitate the handling of evaluation keys.

Note: Here no extensive analysis of function approximations in such compilers is given. In general, FHE allows the evaluation of a functional complete set of operations so the existence of an arbritrary close approximation is given. In general functions like comparison, min or max are most efficiently approximated using circuits and encryption schemes of the third generation. [19] In most other cases a polynomial approximation is feasible.

In concluding this chapter on the limitations of FHE, we have observed that the three primary current constraints of FHE are being progressively mitigated or resolved, as shown in Table 5. This advancement paves the way for an anticipated increase in the adoption of FHE technologies.

Table 5: Main limitations of FHE and their solution

Limitation	potential solution
computational overhead	Hardware acceleration and better packing techniques
lack of standardization	Homomorphic Encryption Standard [4] and stable open source libraries
hard to use	High level compilers like HElayers

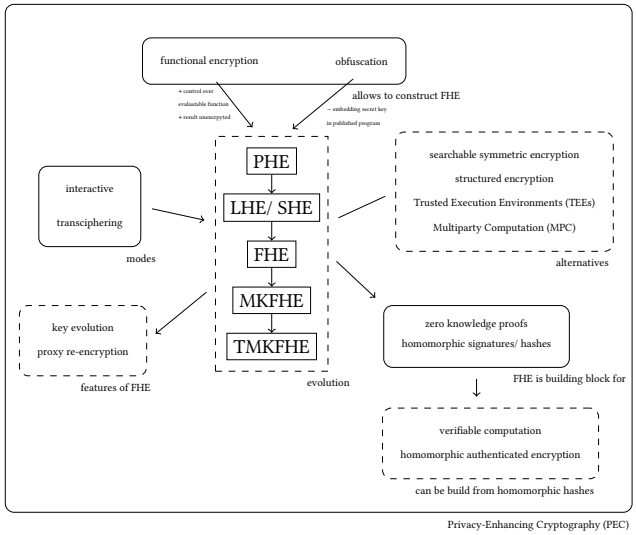

Figure 5: Overview about technqiues related to FHE. FHE stands as a key component in the realm of privacy-enhancing technologies, currently being explored in a dedicated project at the NIST.

2.7 Beyond Homomorphic Encryption

Besides FHE , there are two additional cryptographic concepts that apply functions to data and are intertwined with FHE, namely *Functional Encryption (FE)* and *Obfuscation*.

Definition 2.16 (Functional Encryption (FE)). Possessing a master key MK and a function f, FE enables the derivation of a secret key SK. "Given a ciphertext, the secret key allows the user to learn the value of f applied to the plaintext and nothing else."[6]

[18] It must also be considered whether these approximations introduce any bias into the computational results.

[19] If the min or max is calculated it has to be kept in mind that the encryption is not order-preserving.

The difference between FHE and FE is that FE allows to control which functions can be applied to the data (by the derived secret key that depends on the function f) and a key can be used to get the evaluation result unencrypted.[20] FHE on the other hand allows anybody with the evaluation key to apply any function to the encrypted data and the result is always encrypted. Only the owner of the secret key can get the plaintext result. FE can be used to construct FHE if the functions are applied to encrypted data.

Definition 2.17 (Obfuscation). "Obfuscation was originally designed to be conceptually similar to black box computation, where one gains knowledge of inputs to the black box, and outputs from it, but nothing else." [6]

An obfuscated program incorporating both public and private keys can be created to decrypt the input, apply the necessary function, and then re-encrypt the result. This means one could construct a FHE scheme from obfuscation, but it is important to note that the security implications of embedding secret keys in a published program must be carefully evaluated.

Besides functional encryption and obfuscation, HE can be utilized as a building block for generating zero-knowledge proofs, allowing delegation of computation, or creating homomorphic signatures [6]. Delegation of computation enables the verification of whether an outsourced computation was correctly performed and is closely related to homomorphic signatures, homomorphic hashes, verifiable computation, homomorphic authenticated encryption and homomorphic authentication. Additionally the bootstrapping process in FHE schemes allows key evolution (refer to Theorem 2.15), which can be applied to proxy re-encryption as well [22].

Instead of FHE *searchable symmetric encryption* or *structured encryption* could be used for fast database lookups or data mining in general. Hardware-based solutions like *Trusted Execution Environments (TEE)* could also serve as an alternative to, or in combination with, HE.

TEEs could guarantee the evaluation of the right function or they could be used to perform highly complex operations on intermediate evaluation results unencrypted. The security of such hybrid solutions must be analyzed individually in each case. For instance, a plaintext operation within a Trusted Execution Environment should always be conducted on heavily processed and aggregated data, which do not allow for inference about individual data points.

FHE could also be used to support multi-party computations, where multiple entities evaluate a function on all the input data while keeping their individual inputs private from each other.[21] Multi-key FHE (MKFHE) offers a framework for developing multi-party schemes, yet it necessitates the participation of all parties, as the failure of any single party could compromise the entire protocol. Threshold Multi-key FHE (TMKFHE) provides a more robust solution by tolerating the failure of a limited subset of parties. However, ensuring active security in such a setup requires the incorporation of additional mechanisms, such as Zero-Knowledge Proofs.[22] The exploration of Multi-party FHE extends beyond the scope of this work. In the Homomorphic Encryption Standard [4] TMKFHE is formalized on an abstract level as distributed HE. To gain insight into the practical application of FHE within multi-party use cases, as well as to access additional references, refer to Figure 16 in "Survey on Fully Homomorphic Encryption, Theory, and Applications" by Marcolla et al. [22]. Besides extending FHE to multi-party scenarios, interactive FHE is also conceivable. In interactive FHE, computationally intensive operations (such as bootstrapping) are performed on plaintexts by the client and the resulting encrypted data is sent to the server. While this approach may seem to contradict the principle of outsourcing computation, limiting it to a few operations with exceptionally high overhead can be justifiable. An overview of the mentioned techniques and their relation to FHE is given in Figure 5.

Table 6: Simplified comparison of FHE, MPC and TEE. MPC has a large communication overhead, FHE is computational expensive and TEEs are often proven to be vulnerable against side-channel attacks.

	FHE	MPC	TEE
no communication	●	○	●
no computational overhead	○	●	●
no known attacks	●	●	○
security based on	LWE, RLWE	protocols	hardware

FHE, in conjunction with homomorphic signatures, hardware solutions like trusted execution environments, and zero-knowledge proofs for active security could stand at the forefront of cryptographic solutions for secure cloud usage in a multi-party context. Generally, the future will likely see the combination of various cryptographic techniques with FHE as the optimal way to secure the outsourcing of processing while ensuring the confidentiality of data.

Note: MPC and TEEs are both seen as an alternative to FHE/MKFHE. Each of these technologies offers distinct trade-offs between security, computational overhead, and communication overhead. FHE provides strong data confidentiality with minimal communication needs at the cost of high computational overhead. MPC allows computation on data distributed among multiple parties with strong security guarantees but requires significant communication. TEEs offer an efficient way to securely process data with minimal communication overhead but depend on the security of the hardware implementation and are susceptible to certain types of attacks. A comparison of the three technologies is given in Table 6. The choice between these technologies depends on the specific requirements of the application, including the types of data being processed, the computational resources available, and the threat model considered. The robust security offered by FHE, combined with its adaptability in multi-party scenarios without significant communication overhead, positions it as a highly valuable technology. Its applicability is further enhanced by the ability to offload

[20] "FE is similar in essence to identity-based encryption and attribute-based encryption." [6]

[21] The usage of proxy re-encryption to encrypt the data of all entities under the same new key is not sufficient for multi-party FHE, because it is not resilient to weak collusion attacks between one entity and the proxy.

[22] Passive security means parties follow the protocol and try to extract information. Active security means parties are allowed to deviate from the protocol and send for example wrong shares of information.

the substantial computational demands to potentially more cost-effective, unregulated environments. This flexibility underscores FHE's potential in broadening the scope of secure computational outsourcing.

3 FHE SCHEMES AND IMPLEMENTATIONS

Among the most widely used and implemented SHE schemes are BGV, BFV, FHEW, TFHE, and CKKS.[23]

The BGV, BFV, and GSW schemes are outlined in the Homomorphic Encryption Standard, with CKKS positioned as a prospective addition. Selecting an appropriate scheme is complex, necessitating a nuanced understanding of various factors to optimize performance across different architectures and use cases. A useful starting point for this selection is Table 3 detailing typical characteristics of the schemes of each generation, though it's essential to recognize that actual resource consumption will significantly depend on the specific implementation.

Presently, a multitude of FHE library implementations exist, some of which are highlighted in the Homomorphic Encryption Standard [4] and the survey by Marcolla et al. [22]. These implementations broadly fall into two categories: libraries and compilers/frameworks. FHE libraries primarily provide access to scheme operations through an API, including fundamental functionalities like KeyGen, Enc, Dec, and Eval, alongside additional features for ciphertext management (e.g., noise growth control) and operations such as homomorphic addition and multiplication. The onus of correct API utilization rests on developers, necessitating a deep understanding of each function within privacy-preserving applications.

Conversely, FHE compilers serve as high-level tools that abstract the complexities of FHE library APIs, thereby enabling a broader spectrum of developers to securely implement privacy-preserving mechanisms. These compilers address prevalent engineering challenges in FHE application development, including parameter selection, plaintext encoding, data-independent execution, and ciphertext maintenance. A practical alternative to software optimizations are hardware accelerators that will likely expand the scope of feasible FHE applications.

Given the dynamic nature of implementations, an exhaustive enumeration of libraries and their features is not provided here. Appendix A gives only a short overview of the most important encryption schemes (Table 8), libraries (Table 9), frameworks/compilers (Table 10), and the entities behind their development. An empirical comparison of libraries such as SEAL, Helib, HEAAN, FHEW, TFHE, and Palisade is conducted in the FHEBench study by Jiang and Ju [17], offering valuable insights for selecting suitable schemes and implementations for performing specific operations. Notably, many FHE libraries forego bootstrapping due to its prohibitive computational cost, limiting the number of feasible multiplications and influencing the choice of implementation for particular use cases.

[23]The CKKS scheme, initially termed HEAAN, is now recognized by the research community by the acronym CKKS, which reflects the authors' last names. The designation HEAAN is now applied to the corresponding library [22].

Table 7: Different types of adversaries

type of adversary	behavior
semi-honest/ honest but curious	follows protocol, record data and try to recover information
malicious	most powerful adversary, allowed to deviate from the protocol, inject false data, manipulate data, record data and try to recover information, return false Eval output
covert	malicious adversary with penalties when deviating behavior is detected

4 POSSIBLE USE CASES

FHE allows for a variety of use cases. In general, there are three main types of adversaries that have to be considered in a use-case analysis: semi-honest, malicious and covert. See Table 7 for an overview of adversary types.

Figure 6 presents a comprehensive overview of potential FHE applications in general. The depicted scenarios presuppose a semi-honest adversary model and utilize a non-interactive FHE approach. Should the threat model extend to malicious or covert adversaries, it necessitates the integration of supplementary techniques alongside FHE to ensure robust security measures. The simplest use case is the outsourcing of computation while guaranteeing data confidentiality.

Use Case 1 (Outsourcing Computation). *In this use case, the data owner is also the model owner. Encrypted data, possibly alongside an encrypted model, is sent to the cloud for evaluation. The cloud processes the model on the encrypted data and returns the result, also encrypted.*

This setup ensures data encryption throughout the process (refer to Theorem 2.9), facilitating computation outsourcing to a cost-effective, unregulated environment. Examples include encrypted data database queries or machine learning inference outsourcing, where a pre-trained model[24] is encrypted and stored in the cloud for evaluation on encrypted inputs. The data and the model weights are kept private, but the model structure is exposed.

Use Case 2 (Outsource Data). *In this use case, a data owner transmits their encrypted data to the model owner, who then evaluates the model homomorphically and returns the encrypted results to the data owner.*

This approach enables the assessment of external models on personal data without compromising data confidentiality. A significant benefit is the protection of the model and its structure from exposure. This scenario is particularly appealing to companies

[24]Typically, machine learning models are trained without encryption due to the significant computational overhead associated with FHE.

developing AI models, potentially utilizing AI-specific hardware available to the model owner.

Use Case 3 (Outsource Model). *In this use case, a model owner encrypts and sends their model to a data owner, enabling the model's evaluation on data not owned by the model owner.*

This approach is particularly valuable when encrypted data cannot be shared due to regulatory constraints, and it allows the provision of a trained model to customers without disclosing the model's weights, thus safeguarding training data information. The evaluation occurs on the data owner's side, facilitating decentralized model assessment. Applications include cross-bank model evaluations and government-issued encrypted models for real-time fraud detection on bank data, without revealing model weights.

Use Case 4 (Shared Outsourced Computation I). *In this use case, data and model owners send their encrypted data and model, respectively, to the cloud, outsourcing the entire computation. This could leverage specialized hardware in the cloud and may reduce costs due to cloud competition, maintaining privacy through encryption. It presupposes a non-collusion assumption between the cloud and the secret key holder (in this case the data owner).*

In the event of collusion between the data owner and the cloud, the cloud could transfer the encrypted model to the data owner for decryption with their private key, posing a risk of model exposure. A significant vulnerability in this scenario is the potential for model compromise due to poor key management by only a single data owner (of possible many). The main application of this scenario is the usage of a proprietary AI model of a different company and outsourcing the computation to the cloud. This scenario is primarily intended for employing proprietary AI models from another company and outsourcing computational tasks to the cloud.

Use Case 5 (Shared Outsourced Computation II). *This scenario parallels Use Case 4, with the distinction that the model owner possesses the secret key, necessitating an assumption of non-collusion between the cloud and the model owner.*

This use case facilitates outsourcing both the evaluation process and evaluation on external data. Only the model owner, holding the exclusive knowledge of the secret key, can decrypt the evaluation results. To overcome this limitation controlled randomness (noise) should be added to the plain evaluation results to mitigate the risk of key compromise when results are shared with the data owner.

Use Case 6 (Multi Party Computation). *The previous mentioned use cases can be extended to a multi-party scenario, involving multiple data owners each encrypting their data with a unique key.*

This facilitates the analysis of data across different entities without compromising confidentiality. Such a setup showcases the significant advantages of FHE, particularly its capability to outsource computations on data from diverse sources securely. However, a notable limitation is the further reduction in computational speed compared to standard FHE applications.

To conclude FHE enables leveraging the computational power of the cloud — unregulated and potentially more cost-effective (including the leveraging of specialized hardware) — without compromising the privacy of processed data. FHE facilitates performing any operation on encrypted data, making it ideal for outsourcing computations such as:

- Machine learning (including Neuronal Networks, Random Forests, XGBoost, etc.),
- Basic statistical analyses (such as calculating mean, standard deviation, ARIMA, etc.), and
- Database queries (like set intersections and more).

It additionally facilitates the provision of encrypted models without disclosing the weights (and the architecture in specific scenarios - see Use Case 2). Looking ahead, multi-party FHE is poised to become more feasible, enabling the execution of operations on data from different sources.[25]

Moreover, all the outlined use cases can be executed in an interactive mode, wherein certain operations are conducted on plaintexts by the holder of the secret key, thus optimizing computational resources while maintaining data confidentiality.

Instead of sending FHE-encrypted data, data can be sent to the cloud encrypted with AES, alongside the AES key encrypted with FHE. This provided data enables the cloud to replace AES encryption with FHE encryption, facilitating the execution of homomorphic operations on the encrypted data. Notably, during this encryption substitution, the data remains encrypted at all times (as during bootstrapping). For a detailed illustration of how transciphering operates, refer to Figure 1 in the paper by Aharoni et al. [3].

This technique, known as *Transciphering*, circumvents the need for direct FHE encryption of all data by requiring only AES encryption of the data. This results in less transferred data because the expansion ratio of FHE encryption (2:1) is much higher than with AES encryption (1:1) [3]. The evaluation result is sent back FHE encrypted to the client. Decrypting FHE-encrypted results is generally not very resource-intensive, especially since the evaluation outputs (like classification results or aggregated statistics) are typically small. So Transciphering could allow for significant outsourcing of most of the FHE encryption process.

[25]The bank of the future might securely outsource the complete transaction processing in an encrypted manner to the cloud. Currently, the primary focus of FHE in the financial industry is on conducting data analysis using machine learning on financial data.

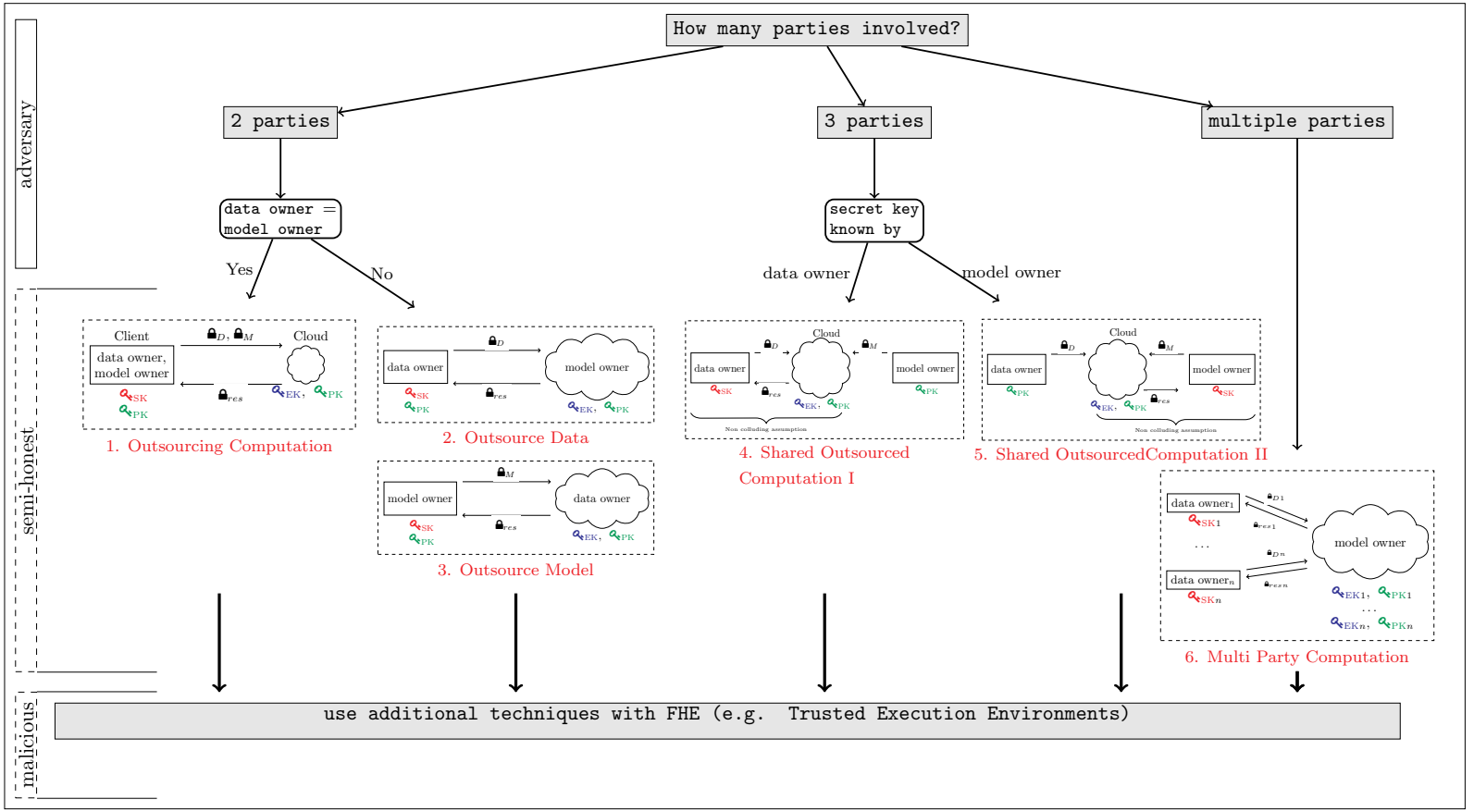

Figure 6: The figure describes FHE use cases, outlining the adversary type in the left column and indicating the computational workload locations with a cloud symbol. A lock symbol specifies what is encrypted and transmitted via FHE: 'M' for Model/Algorithm, 'D' for Data, and 'res' for the evaluation result. The right column suggests possible modes for each use case, including sending data encrypted with AES and converting it securely into FHE encryption in the cloud [3], and performing FHE evaluation in interactive mode to save computing resources by conducting partial computations on plaintexts by the secret key owner [2].

5 CONTRIBUTION

This paper provides foundational knowledge on FHE tailored for beginners. Unlike other surveys on the topic, it emphasizes a practical and theoretical perspective with easy-to-understand explanations, avoiding overly simplified or formally incorrect generalizations.

The main contributions of this paper, distinguishing it from others include:

- A clear distinction between plaintext and ciphertext operations, adding efficiency and security as key attributes of homomorphic encryption schemes.
- A concise delineation of i-hop correctness within FHE, enhancing understanding.
- A streamlined history of FHE, facilitating a comprehensive yet succinct comparison across different generations of FHE.
- The chapter "From SHE to FHE" introduces essential techniques and terms, including all synonymous expressions, which is novel. This aids in better comprehension during further literature review.
- In the "Security" chapter, IND-CPA security is formally defined, and the necessity for noise management is proven. Moreover, the absence of IND-CCA2 security is not only formally demonstrated, but its practical implications are also elucidated.
- The differentiation between key dependent message security and circular security presents a new insight offered by this paper.
- While the issue of incorrect evaluation results was previously recognized, this survey provides an overview of techniques to mitigate such attacks.
- It specifically addresses the need for additional signatures when transferring homomorphically encrypted data, contrasting this with the use of traditional transport encryption.
- A detailed discussion of the limitations of FHE and its positioning within the cryptographic domain represents a novel contribution.

Additionally, providing a comprehensive overview of all potential use cases marks an unprecedented effort in the field.
The comparison between FHE, MPC and TEEs emphasizes the necessity of benchmarking the actual computational overhead associated with FHE. This evaluation constitutes the focus of the subsequent master's thesis.

Future Work

In the master thesis building upon this paper, an empirical analysis (benchmarking) of the CKKS scheme's utility for machine learning using XGBoost on the Bank Marketing dataset is conducted. The primary objective is to predict whether a client will subscribe to a term deposit [23]. The defined use case involves the data owner also being the model owner, with computational tasks outsourced to a server that is semi-honest but not trusted, thus eliminating concerns over incorrect evaluation results in this scenario (see Use Case 1).

To assess the practicality of this setup, two distinct operational modes are examined: the "all-in-one" scenario, where a pre-trained XGBoost model, along with the data, is deployed to the cloud within a docker container, and the "batched" scenario, in which the model is maintained in a docker container on the cloud, with data transmitted via S3 buckets.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2017. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. arXiv:1704.03578 [cs.CR]

[2] Ehud Aharoni, Allon Adir, Moran Baruch, Nir Drucker, Gilad Ezov, Ariel Farkash, Lev Greenberg, Ramy Masalha, Guy Moshkowich, Dov Murik, Hayim Shaul, and Omri Soceanu. 2023. HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data. *Privacy Enhancing Technology Symposium (PETs) 2023* (2023). https://petsymposium.org/2023/paperlist.php,https://github.com/IBM/helayers

[3] Ehud Aharoni, Nir Drucker, Gilad Ezov, Eyal Kushnir, Hayim Shaul, and Omri Soceanu. 2023. E2E near-standard and practical authenticated transciphering. *Cryptology ePrint Archive* (2023).

[4] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard.* Technical Report. HomomorphicEncryption.org, Toronto, Canada.

[5] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. Cryptology ePrint Archive, Paper 2015/046. https://eprint.iacr.org/2015/046 tool: https://github.com/malb/lattice-estimator.

[6] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. 2015. A guide to fully homomorphic encryption. *Cryptology ePrint Archive* (2015).

[7] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. 2018. CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes. Cryptology ePrint Archive, Paper 2018/758. https://eprint.iacr.org/2018/758 https://eprint.iacr.org/2018/758.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption without Bootstrapping. *Electron. Colloquium Comput. Complex.* TR11 (2011). https://api.semanticscholar.org/CorpusID:2182541

[9] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23.* Springer, 409–437.

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. August 2016. TFHE: Fast Fully Homomorphic Encryption Library. https://tfhe.github.io/tfhe/.

[11] Duality. 2023. The HomomorphicEncryption.org Community and the Applied Fully Homomorphic Encryption Standardization Efforts. https://csrc.nist.gov/csrc/media/Presentations/2023/stppa6-fhe/images-media/20230725-stppa6-he-fhe--kurt-rohloff.pdf. Accessed: 2024-01-29.

[12] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques.* Springer, 617–640.

[13] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[14] Craig Gentry. 2009. *A fully homomorphic encryption scheme.* Stanford university.

[15] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I.* Springer, 75–92.

[16] Shai Halevi. 2017. *Homomorphic Encryption.* Springer International Publishing, Cham, 219–276. https://doi.org/10.1007/978-3-319-57048-8_5

[17] Lei Jiang and Lei Ju. 2022. Fhebench: Benchmarking fully homomorphic encryption schemes. *arXiv preprint arXiv:2203.00728* (2022).

[18] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. 2021. Revisiting Homomorphic Encryption Schemes for Finite Fields. In *Advances in Cryptology – ASIACRYPT 2021*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer International Publishing, Cham, 608–639.

[19] Çetin Kaya Koç, Funda Özdemir, and Zeynep Ödemiş Özger. 2021. *Partially Homomorphic Encryption.* Springer International Publishing. https://doi.org/10.1007/978-3-030-87629-6

[20] Baiyu Li and Daniele Micciancio. 2021. On the security of homomorphic encryption on approximate numbers. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40.* Springer, 648–677.

[21] Tal Malkin, Isamu Teranishi, and Moti Yung. 2011. Key dependent message security: recent results and applications. In *Proceedings of the first ACM conference on Data and application security and privacy*. 3–12.
[22] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H.P. Fitzek, and Najwa Aaraj. 2022. Survey on Fully Homomorphic Encryption, Theory, and Applications. Cryptology ePrint Archive, Paper 2022/1602. https://doi.org/10.1109/JPROC.2022.3205665 https://eprint.iacr.org/2022/1602.
[23] S. Moro, P. Rita, and P. Cortez. 2012. Bank Marketing. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5K306.
[24] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. 1978. On data banks and privacy homomorphisms. *Foundations of secure computation* 4, 11 (1978), 169–180.
[25] VF Rocha, Julio López, and V Falcão Da Rocha. 2018. An overview on homomorphic encryption algorithms. *UNICAMP Universidade Estadual de Campinas, Tech. Rep* (2018).
[26] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. 2010. Fully homomorphic encryption over the integers. In *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*. Springer, 24–43.

DECLARATION OF INDEPENDENCE

Hereby, I, Felix Peter Paul (Matriculation No.: 2738210), declare that I have independently composed the present work and have not used any sources or aids (including electronic media and online sources) other than those indicated. I am aware that an attempt at deception or a breach of regulations exists if this declaration proves to be untrue.

Place, Date | Signature

A ADDITIONAL INFORMATION

Most active companies in FHE [4, 22]

- IBM
- Microsoft
- Duality Technologies
- Zama AI
- Cryptolab
- Google
- Intel

B ADDITIONAL NOTES

Runtime comparison of FHE vs. plaintext computation [11]
Often, the runtime of FHE is compared to plaintext runtimes. A realistic goal for the future is to achieve homomorphic multiplication on dedicated hardware at speeds comparable to conventional multiplication in software on a CPU. However, in some scenarios, comparing with plaintext computations is not meaningful. For instance, FHE enables financial crime investigation on encrypted data within minutes, a process that would typically require a judicial order, taking days to obtain. Furthermore, FHE opens up unprecedented possibilities. For example, Threshold Multi-key FHE is expected to facilitate the practical evaluation of data from various, mutually distrustful sources in the future.

FHE vs. use case specific solutions
Sometimes, solutions specific to particular use cases may be more practical than FHE. The main issue with highly specialized solutions is often the lack of in-house and industry-wide expertise, along with the absence of standardization. Nonetheless, before deploying FHE, the use of well-known techniques should always be evaluated. For instance, salt could be utilized to compute private set intersections.

Intellectual Property Rights [11]
The Intellectual Property Rights (IPR) situation requires clarification. IBM possesses a broad patent on FHE, while Seoul National University has a patent on CKKS schemes. However, the validity and scope of these patents remain uncertain. So the selection of an encryption scheme has to be influenced by considerations related to Intellectual Property Rights.

Table 8: Most important HE schemes [17]

Operation	BFV [13]	BGV [8]	CKKS [9]	FHEW [12]	TFHE [10]
Native Add/Sub	●	●	●	○	○
Native Mult	●	●	●	○	○
SIMD	●	●	●	(●)	(●)
Boolean Logic	○	●	○	●	●
$< 1s$ Bootstrapping	○	○	○	●	●

Table 9: Most important HE libraries [4, 22]

	Library	Language	Schemes				
			BGV	BFV	FHEW	TFHE	CKKS
in HeLayers	HEAAN	C++	○	○	○	○	●
	HElib	C++	●	○	○	○	●
	PALISADE	C++	●	●	●	●	●
	OpenFHE	C++	●	●	●	●	●
	Lattigo	Go	●	●	○	○	●
	SEAL	C++/ C#	●	●	○	○	●
	FHEW	C++	○	○	●	○	○
	TFHE	C++/ C	○	○	○	●	○
	concrete	Rust	○	○	○	●	○
	RNS-HEAAN	C++	○	○	○	○	●
	FV-NFLlib	C++	○	●	○	○	○
	CuFHE	Cuda/C++	○	○	○	●	○
	NuFHE	Python	○	○	○	●	○

Table 10: Most important HE compilers [22]

Compiler	Language	Library					
		HElib	SEAL	PALISADE	FHEW	TFHE	HEAAN
ALCHEMY	Haskell	○	○	○	○	○	○
Cingulata	C++	○	○	○	○	●	○
E^3	C++	●	●	●	●	●	○
SHEEP	C++	●	●	●	○	●	○
EVA	C++	○	●	○	○	○	○
Marble	C++	●	●	○	○	○	○
RAMPARTS	Julia	○	○	●	○	○	○
Transpiler	C++	○	○	●	○	●	○
CHET	C++	○	●	○	○	○	●
nGraph-HE	C++	○	●	○	○	○	○
SEALion	C++	○	●	○	○	○	○
HElayers	C++, python API	●	●	●	○	○	●

Table 11: Homomorphic properties of well-known PHE schemes [1]

		Homomorphic Operation			
Scheme	Year	Add	Mult	security	note
RSA	1978		✓	factoring	
Goldwasser and Micali	1982	✓		quadratic residuosity	first probabilistic
El-Gamal	1985		✓	discrete logarithm	
Benaloh	1994	✓		higher residuosity	
Naccache and Stern	1998	✓		composite residuosity	extends benaloh
Okamoto and Uchiyama	1998	✓		p-subgroup assumption	improves performance of old schemes
Paillier	1999	✓		composite residuosity	probablistic
Damgård and Jurik	2001	✓			generalize Paillier
Galbraith	2002	✓		elliptic curves	generalize Paillier
Kawachi et al.	2007	✓		lattice problems	