A Guide to Fully Homomorphic Encryption

Frederik Armknecht[1], Colin Boyd[2], Christopher Carr[2], Kristian Gjøsteen[3], Angela Jäschke[1], Christian A. Reuter[1], and Martin Strand[3]

[1]University of Mannheim
{armknecht, jaeschke, reuter}@uni-mannheim.de
[2]Department of Telematics, NTNU
{colin.boyd, ccarr}@item.ntnu.no
[3]Department of Mathematical Sciences, NTNU
{kristian.gjosteen, martin.strand}@math.ntnu.no

Abstract

Fully homomorphic encryption (FHE) has been dubbed the holy grail of cryptography, an elusive goal which could solve the IT world's problems of security and trust. Research in the area exploded after 2009 when Craig Gentry showed that FHE can be realised in principle. Since that time considerable progress has been made in finding more practical and more efficient solutions. Whilst research quickly developed, terminology and concepts became diverse and confusing so that today it can be difficult to understand what the achievements of different works actually are. The purpose of this paper is to address three fundamental questions: What is FHE? What can FHE be used for? What is the state of FHE today? As well as surveying the field, we clarify different terminology in use and prove connections between different FHE notions.

1 Introduction

The purpose of homomorphic encryption is to allow computation on encrypted data. Thus data can remain confidential while it is processed, enabling useful tasks to be accomplished with data residing in untrusted environments. In a world of distributed computation and heterogeneous networking this is a hugely valuable capability. Finding a general method for computing on encrypted data had been a goal in cryptography since it was proposed in 1978 by Rivest, Adleman and Dertouzos [54]. Interest in this topic is due to its numerous applications

in the real world. The development of fully homomorphic encryption is a revolutionary advance, greatly extending the scope of the computations which can be applied to process encrypted data homomorphically. Since Gentry published his idea in 2009 [28, 29] there has been huge interest in the area, with regard to improving the schemes, implementing them and applying them.

We look in detail at specific applications in Section 2, but to give a feeling, consider cloud computing. As more and more data is outsourced into cloud storage, often unencrypted, considerable trust is required in the cloud storage providers. The Cloud Security Alliance lists data breach as the top threat to cloud security [61]. Encrypting the data with conventional encryption avoids the problem. However, now the user cannot operate on the data and must download the data to perform the computations locally. With fully homomorphic encryption the cloud can perform computations on behalf of the user and return only the encrypted result.

1.1 What is Fully Homomorphic Encryption?

Principally, FHE allows for arbitrary computations on encrypted data. Computing on encrypted data means that if a user has a function f and want to obtain $f(m_1, \ldots, m_n)$ for some inputs $m_1, \ldots, m_n$, it is possible to instead compute on encryptions of these inputs, $c_1, \ldots, c_n$, obtaining a result which decrypts to $f(m_1, \ldots, m_n)$.

In some cryptosystems the input messages (plaintexts) lie within some algebraic structure, often a group or a ring. In such cases the ciphertexts will often also lie within some related structure, which could be the same as that of the plaintexts. The function f in older homomorphic encryption schemes is typically restricted to be an algebraic operation associated with the structure of the plaintexts. For instance, consider ElGamal. If the plaintext space is a group G, then the ciphertext space is the product $G \times G$, and f is restricted to the group operation on G. Indeed most schemes prior to 2009 fit such a structure. We can express the aim of fully homomorphic encryption to be to extend the function f to be *any function*. This aim can be achieved if the scheme is homomorphic with respect to a functionally complete set of operations and it is possible to iterate operations from that set.

While it is always a requirement that encryption schemes are efficient in a theoretical sense, namely running in polynomial time in the security parameter, practical efficiency was not the first priority in obtaining the first FHE schemes. One reason for the lack of efficiency of these schemes is that they use a plaintext space consisting of a single bit and are homomorphic with respect to addition and multiplication modulo 2. While any function of any complexity can be built up from such basic operations, that may require a large number of such operations.

In order to move towards better efficiency, some recent variants of FHE schemes restrict the functions f in different ways which we will explore later.

Although a theoretical view of FHE cares only about maximising the choices of f, a practical view cares also about keeping this choice only as large as needed,

and may also prefer a richer structure for the plaintext and ciphertext spaces that just the binary case.

1.2 Relations of FHE: Functional Encryption and Program Obfuscation

The fundamental idea behind FHE is to be able to apply functions on encrypted data. Two other cryptographic notions formed with functions in mind are *functional encryption* and *obfuscation*. Intriguingly, obfuscation, functional encryption and fully homomorphic encryption seem somehow intertwined, as has been previously recognised [3, 25].

Functional encryption (FE) is similar in essence to identity based encryption and attribute based encryption. Boneh, Sahai and Waters [13] give a concise explanation of the relations between these three notions, as well as some discussion on FHE. The concepts of FHE and FE do indeed have some overlap, and it has been demonstrated that functional encryption can work as FHE, with some slight adaptation [3].

Functional encryption allows a secret key to be issued using a master key, dependent on a function f. Given a ciphertext, the secret key allows the user to learn the value of f applied to the plaintext and nothing else [12]. Computing functions on encrypted data links the two concepts of homomorphic and functional encryption. A notable difference is the way the functions are applied. FE grants control over what functions can be applied to the data via a master key holder, who issues keys based on a decision of the appropriateness of the function. A key can be used to obtain the plaintext result of the function applied on the encrypted data. FHE permits functions to be the run by anyone with the *evaluation key* (see Section 3), however only the owner of the secret key can decrypt the result into plaintext. The user running the function only gets ciphertext.

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 [9, 38, 25]. With obfuscation, one could place keys within the program to be run without revealing knowledge of the keys. One could thus generate an obfuscated program that contains the public and private keys, and process the input by applying first the decryption algorithm, next the required function, and finally encrypting the result. This would act as a replacement for the homomorphic operation in FHE.

This ability to generate a FHE scheme from an obfuscation scheme and a traditional encryption scheme may seem promising, but practically it remains unclear if this offers an advantage over direct FHE. We would also need to consider the security constraints and implications of *hiding* secret keys inside a published program.

1.3 Need for Systematization

Treatment of FHE can seem very confusing. Sometimes, two definitions seem to say the same thing – for example, at first glance, being able to evaluate an arbitrary circuit and being able to evaluate arbitrarily many circuits consecutively seems to be the same thing. This, however, is not the case as will be explained in Remark 5.

To help understand the distinction, consider the cloud computing example: FHE is usually sold as the solution. However, if we can only evaluate one circuit of arbitrary size, then we cannot use intermediate results for further computations later; everything has to be computed from scratch through the original ciphertexts. This satisfies the usual definition of FHE (Definition 9), but is unintuitive and is hardly an optimal solution. What is needed in this scenario is the ability to evaluate arbitrarily many circuits consecutively.

This highlights another problem in this field: in some cases, definitions do not express what one would intuitively assume. In other cases, one intuition has different definitions in different papers. This, for example, is the case for an attribute called *compactness*, which intuitively says that the ciphertext size should not be growing through homomorphic operations. Gentry defines it through one characterization in his original work, while in subsequent works a different characterization is used. Seeing that both definitions are equivalent is not as straightforward as one may assume, and actually requires an additional assumption.

Sometimes, attributes are not properly defined at all, and sometimes implications are used that have not been mentioned in the same paper. Figure 1 gives an idea of how complex this jungle of definitions is. Starting with the definitions and properties (white rectangles in the figure) we can give classifications of the different kinds of homomorphic schemes (shaded round rectangles). Furthermore, these classifications can again be combined with hop correctness, which yields another set of homomorphic schemes (darker oval shapes).

1.4 Our Contribution

First, in Section 2, we gather existing applications of (fully) homomorphic encryption mentioned in the literature, examine their usefulness both in practice and as building blocks for other cryptographic schemes and point out their limitations.

Next, we provide the much needed organization of terminology. We present existing definitions in a consistent way, reconciling different definitions for one notion when they exist, and explaining points of potential confusion. Furthermore, we introduce new definitions, enabling a better understanding of existing schemes and existing definitions. This contribution constitutes Section 3 and is summarised in Figure 1. In Section 4, we formally prove some elementary relations between notions presented in Section 3. To give an overview of the current landscape in FHE research, Section 5 summarises some existing schemes along with the underlying hardness assumptions and runtimes where available.

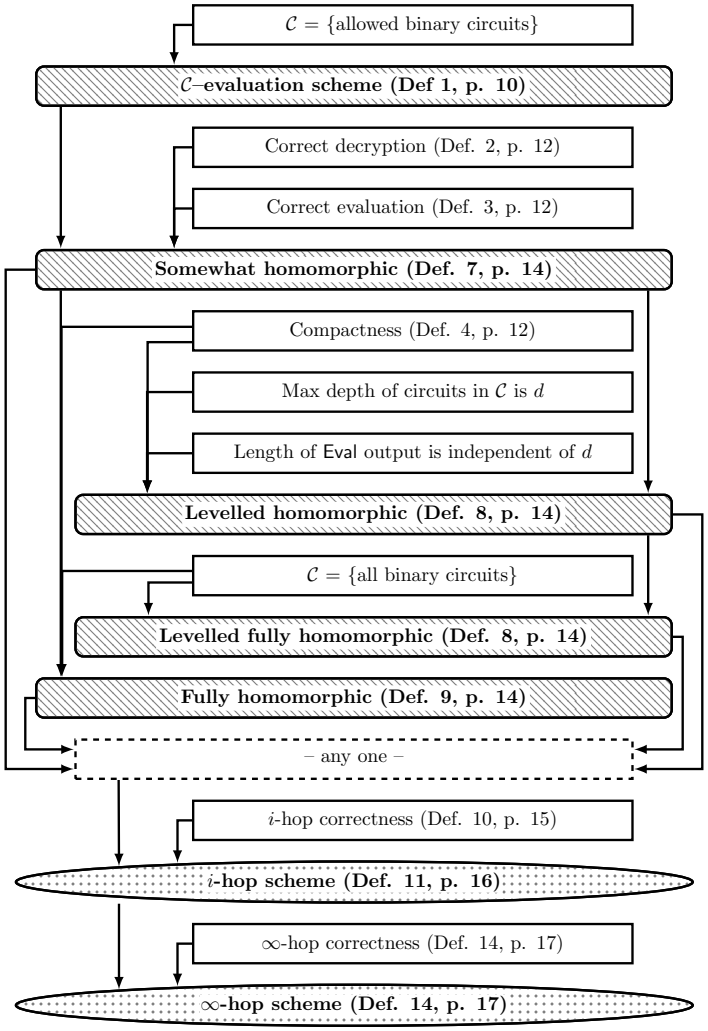

Figure 1: Classifying FHE. The definitions are white rectangles. The classes are shaded round rectangles. The round shapes below represent the add-on properties of hops. The arrows between the elements of the graph show their dependencies. The integer d specifies the maximum depth of circuits in the set of allowed binary circuits $\mathcal{C}$.

2 Applications of FHE

This section explores the numerous applications of the various flavours of homomorphic encryption. Some require *fully* homomorphic encryption, while others just need *somewhat* homomorphic encryption. The distinction will become clear in Section 3. For now, it suffices to know that a fully homomorphic scheme can compute anything on encrypted data, while a somewhat homomorphic scheme is more restricted.

This section is divided into three parts. The first part deals with applications that are feasible today, the second examines constructions that use homomorphic encryption as building blocks, and the third looks at current limitations of FHE.

2.1 Practical Applications of FHE

Although still slow (see Section 5), homomorphic encryption has been proposed for several practical uses. This section lists those applications that are conceivable with the technology we have today.

2.1.1 Consumer Privacy in Advertising

Though often unwanted, advertising can be useful when tailored to user needs, e.g. through recommender systems or through location-based advertising. However, many users are concerned about the privacy of their data, in this case their preferences or location. There have been several approaches to this problem.

Jeckmans et al. [43] sketch a scenario where a user wants recommendations for a product. The scenario is designed around a social network where recommendations are based on the tastes of the user's friends with the condition of confidentiality. The proposed system applies homomorphic encryption to allow a user to obtain recommendations from friends without the identity of the recommender being revealed.

Armknecht and Strufe [6] presented a recommender system where a user gets encrypted recommendations without the system being aware of the content. This system builds upon a very simple but highly efficient homomorphic encryption scheme which has been developed for this purpose. This allows a function to be computed which chooses the advertisement for each user while the advertising remains encrypted.

In another approach to personalized advertising [50] a mobile device sends a user's location to a provider, who sends customized ads, such as discount vouchers for nearby shops, back to the user. Of course, this potentially allows the provider to monitor everything about the user's habits and preferences. However, this problem can be solved by homomorphic encryption – provided the advertisements come from a third party (or several) and there is no collusion with the provider.

2.1.2 Medical Applications

Naehrig et al. [50] propose a scenario where a patient's medical data is (continuously) uploaded to a service provider in encrypted form. Here, the user is the data owner, so the data is encrypted under the user's public key and only the user can decrypt. The service provider then computes on the encrypted data, which could consist of things like blood pressure, heart rate, weight or blood sugar reading to predict the likelihood of certain conditions occurring or more generally to just keep track of the user's health. The main benefit here is to allow real-time health analysis based on readings from various sources without having to disclose this data to any one source. Lauter [45] described an actual implementation of a heart attack prediction by Microsoft.

2.1.3 Data Mining

Mining from large data sets offers great value, but the price for this is the user's privacy. While Yang, Zhong and Wright [64] are often cited as using homomorphic encryption as a solution to this problem, the scheme actually uses *functional encryption*, a common confusion discussed in Section 1.2. However, applying homomorphic encryption is certainly conceivable as a solution.

2.1.4 Financial Privacy

Imagine a scenario where a corporation has sensitive data and also proprietary algorithms that they do not want disclosed, e.g. stock price prediction algorithms in the financial sector. Naehrig et al. [50] propose the use of homomorphic encryption to upload both the data and the algorithm in encrypted form in order to outsource the computations to a cloud service.

However, keeping the algorithm secret is not something that homomorphic encryption offers, but is rather part of *obfuscation* research (see section 1.2). The attribute that comes closest in fully homomorphic schemes is called *circuit privacy*, but this merely guarantees that no information about the function is leaked by the output – not that one can encrypt the function itself.

What homomorphic encryption offers is the solution to a related problem. Imagine that a corporation A has sensitive data, like a stock portfolio, and another company B has secret algorithms that make predictions about the stock price. If A would like to use B's algorithms (for a price, of course), either A would have to disclose the stock portfolio to B, or B has to give the algorithm to A. With homomorphic encryption, however, A can encrypt the data with a circuit private scheme and send it to B, who runs the proprietary algorithm and only sends back the result, which can only be decrypted by A's secret key. This way, B does not learn anything about A's data, and A does not learn anything about the algorithms used.

2.1.5 Forensic Image Recognition

Bösch et al. [14] describe how to outsource forensic image recognition. Tools similar to this are being used by the police and other law enforcement agencies to detect illegal images in a hard drive, network data streams and other data sets. The police use a database containing hash values of "bad" pictures. in. A major concern is that perpetrators could obtain this database, check if their images would be detected and, if so, change them.

This scheme uses a somewhat homomorphic encryption scheme proposed by Brakerski and Vaikuntanathan [17] to realise a scenario where the police database is encrypted while at the same time the company's legitimate network traffic stays private. The company compares the hashed and encrypted picture data stream with the encrypted database created by the police. The service provider learns nothing about the encrypted database itself, and after a given time interval or threshold, the temporary variable is sent to the police.

2.2 Homomorphic Encryption Schemes as Building Blocks

Homomorphic encryption schemes can be used to construct cryptographic tools such as zero knowledge proofs, signatures, MACs and multiparty computation implementations.

2.2.1 Zero Knowledge Proofs

Gentry shows in his dissertation [28] that homomorphic encryption can be used in the construction of non-interactive zero knowledge (NIZK) proofs of small size. A user wants to prove knowledge of a satisfying assignment of bits $\pi_1, \ldots, \pi_t$ for a boolean circuit C. The NIZK proof consists of generating a public key, encrypting the π_i's and homomorphically evaluating C on these encryptions. A standard NIZK proof is attached to prove that each ciphertext encrypts either 0 or 1 and that the output of the evaluation encrypts 1.

2.2.2 Delegation of Computation

Outsourcing computation is the second big pillar in cloud computing, besides outsourcing data. A user may want to delegate the computation of a function f to the server. However, the server may be malicious or just prone to malfunctions, meaning the user may not trust the result of the computation. The user wants to have a proof that the computation was done correctly and verifying this proof should also be significantly more efficient than the user doing the computation.

Chung et al. [18] use fully homomorphic encryption to design schemes for delegating computation, improving the results of Gennaro et al. [26], while van Dijk and Juels [63] examine the infeasibility of FHE alone solving privacy issues in cloud computing.

One example for the delegation of computation is *message authenticators*. A user who has outsourced computation on a data set might want to check that the

return value is really the correct result. The tag should be independent of the size of the original data set, and only verifiable for the holder of the private key. Gennaro and Wichs [27] propose such a scheme based on a fully homomorphic encryption scheme, which can be considered as a symmetric-key version of fully homomorphic signatures [10]. However, it only supports a bounded number of verification queries.

2.2.3 Signatures

Gorbunov et al. [39] presented a construction of levelled fully homomorphic signature schemes. The scheme can evaluate arbitrary circuits with maximal depth d over signed data and homomorphically produce a short signature which can be verified by anybody using the public verification key. The user uploads the signed data x, then the server runs some function g over the data which yields $y = g(x)$. Additionally, the server publishes the signature $\sigma_{g,y}$ to verify the computation.

This work also introduces the notion of *homomorphic trapdoor functions* (HTDF), one of the building blocks for the signature construction. HTDF themselves are based on the small integer solution (SIS) problem. The first definition of fully homomorphic signatures was given in Boneh and Freeman [10].

2.2.4 Multiparty Computation

Multiparty computation requires interaction between participants. Damgård et al. [21] provide a description of how a somewhat homomorphic scheme can be used to construct offline multiplication during the computations. The players use the somewhat homomorphic scheme in a preprocessing phase, but return to the much more efficient techniques of multiparty computation in the computation phase.

2.3 Limitations of FHE

Both in literature and intuitively, there are several applications which permit fully homomorphic encryption as a solution. However, in this subsection, we discuss three main limitations of FHE in real-world scenarios. Afterwards, we present some examples, all of which contain the first two limitations.

The first limitation is support for multiple users. Suppose there are many users of the same system (which relies on an internal database that is used in computations), and who wish to protect their personal data from the provider. One solution would be for the provider to have a separate database for every user, encrypted under that user's public key. If this database is very large and there are many users, this would quickly become infeasible. López-Alt et al. [46] have shown promising directions to address this problem by defining and constructing multi-key FHE.

Next, there are limitations for applications that involve running very large and complex algorithms homomorphically. All fully homomorphic encryption schemes today have a large computational overhead, which describes the ratio of computation time in the encrypted version versus computation time in the clear. Although polynomial in size, this overhead tends to be a rather large polynomial, which increases runtimes substantially and makes homomorphic computation of complex functions impractical. Even if in the future an extremely efficient FHE should be found, other problems remain. For example, for circuits, there is no concept of aborting an algorithm when operating on encrypted data. In the case of comparison, this would require to run the full circuit which is large by itself. In other words, certain mechanisms seems to get significantly more involved just because values remain hidden. One way to solve this problem is suggested by Goldwasser et al. [37] by using Turing machines instead of circuits.

Finally, FHE does not necessarily imply secret function evaluation. We already encountered this in the discussion of the applicability to financial data above. This issue belongs to the research on obfuscation.

3 Definitions

This section gives an overview of the terminology used in the literature on FHE. Some of our definitions come directly from existing papers while others have been rephrased, either because there were no satisfactory formal definitions or to fit the definitions into our formal framework; we give citations in the first case.

We begin with a space $\mathcal{P} = \{0, 1\}$, which we call the plaintext space, and a family F of functions from tuples of plaintexts to $\mathcal{P}$. We can express such a function as a Boolean circuit on its inputs. If we denote this circuit by C, we use ordinary function notation $C(m_1, m_2, \ldots, m_n)$ to denote the evaluation of the circuit on the tuple $(m_1, m_2, \ldots, m_n)$. Our first definition follows Brakerski and Vaikuntanathan [16].

Definition 1 ($\mathcal{C}$–Evaluation Scheme)**.** Let $\mathcal{C}$ be a set of circuits. A $\mathcal{C}$–evaluation scheme for $\mathcal{C}$ is a tuple of probabilistic polynomial–time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ such that:

$\mathsf{Gen}(1^\lambda, \alpha)$ is the key generation algorithm. It takes two inputs, security parameter λ and auxiliary input α, and outputs a key triple (pk, sk, evk), where pk is the key used for encryption, sk is the key used for decryption and evk is the key used for evaluation.

$\mathsf{Enc}(pk, m)$ is the encryption algorithm. As input it takes the encryption key pk and a plaintext m. Its output is a ciphertext c.

$\mathsf{Eval}(evk, C, c_1, \ldots, c_n)$ is the evaluation algorithm. It takes as inputs the evaluation key evk, a circuit $C \in \mathcal{C}$ and a tuple of inputs that can be a mix of ciphertexts and previous evaluation results. It produces an evaluation output.

$\mathsf{Dec}(sk, c)$ is the decryption algorithm. It takes as input the decryption key sk and either a ciphertext or an evaluation output and produces a plaintext m.

Here $\mathcal{X}$ denotes the ciphertext space which contains the *fresh ciphertexts* (see equation (1)), $\mathcal{Y}$ denotes the space of evaluation outputs and $\mathcal{Z}$ is the union of both $\mathcal{X}$ and $\mathcal{Y}$. $\mathcal{Z}^*$ contains arbitrary length tuples made up of elements in $\mathcal{Z}$. The key spaces are denoted by $\mathcal{K}_p, \mathcal{K}_s$ and $\mathcal{K}_e$, respectively, for pk, sk and evk. The public key contains a description of the plaintext and ciphertext spaces. The input to the key generation algorithm Gen is given in unary notation, i.e., 1^λ. Gen may also take another optional input α from the space $auxs$, this is the auxiliary input and will become clear in Remark 3. Finally, $\mathcal{C}$ is the set of *permitted circuits*, i.e. all the circuits which the scheme can evaluate.

With these spaces defined, the domain and range of the algorithms are given by

$$\begin{aligned} \mathsf{Gen}: &\quad \mathbb{N} \times \mathcal{A} \to \mathcal{K}_p \times \mathcal{K}_s \times \mathcal{K}_e \\ \mathsf{Enc}: &\quad \mathcal{K}_p \times \mathcal{P} \to \mathcal{X} \\ \mathsf{Dec}: &\quad \mathcal{K}_s \times \mathcal{Z} \to \mathcal{P} \\ \mathsf{Eval}: &\quad \mathcal{K}_e \times \mathcal{C} \times \mathcal{Z}^* \to \mathcal{Y} \end{aligned}$$

where $\mathcal{X} \cup \mathcal{Y} = \mathcal{Z}$ and $\mathcal{A}$ is an auxiliary space. Note that in general the evaluation space can be disjoint from the ciphertext space.

Throughout this paper, we treat the ciphertext space $\mathcal{X}$ as the image of encryption, and the evaluation space $\mathcal{Y}$ as the image of evaluation. Therefore $\mathcal{Z}$ cannot contain an element that is not a possible output of the encryption algorithm or the evaluation algorithm. Formally,

$$\mathcal{X} = \{c \mid \Pr[\mathsf{Enc}(pk, m) = c] > 0, m \in \mathcal{P}\} \tag{1}$$

and

$$\mathcal{Y} = \{z \mid \Pr[\mathsf{Eval}(evk, C, c_1, \ldots, c_n) = z] > 0, c_i \in \mathcal{Z}, \text{ and } C \in \mathcal{C}\}.$$

Notably, the evaluation key is often also part of the public key. By defining the scheme this way, with a separate evaluation key, we are not forbidding $pk = evk$ but asserting that it is not strictly necessary. Separate pk and evk is becoming a standard definition [16, § 3.1].

Remark 1 (Ciphertext decryption). Brakerski and Vaikuntanathan [17] mention that running the decryption algorithm on an output of the encryption algorithm is not strictly necessary: "... we do not require that the ciphertexts c_i are decryptable themselves, only that they become decryptable after homomorphic evaluation." They point out that one can always evaluate the encrypted ciphertext with a *blank circuit* (essentially a circuit computing the function $f(x) = x$) before decryption, thus simplifying the allowed inputs to the decryption algorithm. From now on, we allow the decryption of fresh ciphertexts, as this seems

a more natural approach and applies to most known FHE schemes. The decryption algorithm can operate on ciphertexts or evaluations (take values from both the ciphertext space and the evaluation space). This choice removes the need for a blank circuit. In general though, this distinction is not necessary, especially when the evaluation space and the ciphertext space are the same.

3.1 Attributes

Presented here are the attributes of homomorphic encryption schemes. On the one hand, we need things like correctness to even call this an encryption scheme, and on the other hand we define attributes like compactness and circuit privacy which exclude trivial solutions to the problem of homomorphic encryption.

Definition 2 (Correct Decryption)**.** A $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is said to *correctly decrypt* if for all $m \in \mathcal{P}$,

$$\Pr[\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m] = 1,$$

where sk and pk are outputs of $\mathsf{Gen}(1^\lambda, \alpha)$.

This means that we must be able to decrypt a ciphertext to the correct plaintext, without error.

Definition 3 (Correct Evaluation, [16, Def. 3.3])**.** A $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) *correctly evaluates* all circuits in $\mathcal{C}$ if for all $c_i \in \mathcal{X}$, where $m_i \leftarrow \mathsf{Dec}(sk, c_i)$, for every $C \in \mathcal{C}$, and some negligible function ϵ,

$$\Pr[\mathsf{Dec}(sk, \mathsf{Eval}(evk, C, c_1, \ldots, c_n)) = C(m_1, \ldots, m_n)] = 1 - \varepsilon(\lambda)$$

where sk, pk and evk are outputs of $\mathsf{Gen}(1^\lambda, \alpha)$.

This means that with overwhelming probability, decryption of the homomorphic evaluation of a permitted circuit yields the correct result. Note that for Definition 2 and 3 we are intentionally restricting to $\mathcal{X}$ and not to $\mathcal{Y}$. This is developed further in Section 3.3.

From now on, we say a $\mathcal{C}$–evaluation scheme is *correct* if it has the properties of both correct evaluation and correct decryption.

Definition 4 (Compactness [62, Def. 3])**.** A $\mathcal{C}$–evaluation scheme is *compact* if there is a polynomial p, such that for any key-triple (sk, pk, evk) output by $\mathsf{Gen}(1^\lambda, \alpha)$, any circuit $C \in \mathcal{C}$ and all ciphertexts $c_i \in \mathcal{X}$, the size of the output $\mathsf{Eval}(evk, C, c_1, \ldots, c_n)$ is not more than $p(\lambda)$ bits, independent of the size of the circuit.

This means that the ciphertext size does not grow much through homomorphic operations and the output length only depends on the security parameter. This also rules out the trivial homomorphic scheme where the evaluation algorithm is the identity function (that is, it outputs $(C, c_1, \ldots, c_n)$), and the decryption function is defined to decrypt the input ciphertexts $c_1, \ldots, c_n$, apply the appropriate function to the corresponding plaintexts, and output this result [30].

Remark 2 (On compactness). Gentry's original definition was a slightly different one, which could informally be paraphrased as: *The scheme is compact if there exists a circuit C_D of "reasonable" length that computes the decryption circuit.* This definition relies on the size of the decryption circuit. However, we feel that the first definition, which relies on the length of Eval's output – given intuitively in his work – and used as the definition of compactness in following works [16, 62], provides for a better understanding. We further examine the relationship between these two concepts (and state the latter one formally) in Section 4.1.

In anticipation of following results, we introduce another definition, originally used by Gentry, that groups all of the definitions seen so far in this section [28, Def 2.1.2].

Definition 5 (Compactly Evaluate)**.** A $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) *compactly evaluates* all circuits in $\mathcal{C}$ if the scheme is compact and correct.

We now define circuit privacy. One may easily confuse circuit privacy semantically with *circuit obfuscation*, because both seem to keep the circuit secret or private. However, circuit obfuscation deals with the concealing of the circuit. This is important if the used algorithms themselves are valuable and ought to be secret. In contrast, circuit privacy characterizes the distributions of the output of the algorithms Eval and Enc.

Definition 6 (Circuit Privacy [28, Def. 2.1.6])**.** A $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is said to be *perfectly/statistically/computationally circuit private* if for any key-triple (sk, pk, evk) output by $\mathsf{Gen}(1^\lambda, \alpha)$, for all circuits $C \in \mathcal{C}$ and all $c_i \in \mathcal{X}$, such that $m_i \leftarrow \mathsf{Dec}(sk, c_i)$, the two distributions on $\mathcal{Z}$

$$D_1 = \mathsf{Eval}(evk, C, c_1, \ldots, c_n)$$

and

$$D_2 = \mathsf{Enc}(pk, C(m_1, \ldots, m_n)),$$

both taken over the randomness of each algorithm, are *perfectly*, *statistically* or *computationally indistinguishable*, respectively.

Why this definition implies that the circuit is *private* may not be immediately clear. Essentially, it states that the output from the evaluation of a specific circuit on ciphertexts looks like the output from the encryption of a plaintext value v, generated in this case by the circuit and the corresponding plaintexts. As v is just another plaintext (i.e. $v = C(m_1, m_2, \ldots, m_n)$), it is *difficult* for determine how it was generated (the level of difficulty is hierarchical from perfect to computational).

Circuit privacy has also been known by the name *strongly homomorphic* [19, 56] in the literature, and there still remains a slight point of divergence within the community on the accurate definition of circuit privacy. Whilst we keep the original definition as given by Gentry [28], a slightly weaker notion exists that is similar, namely *function privacy*. The important difference is that

function privacy only requires that evaluating different circuits on encrypted data produces distributions that are statistically close, computationally close or identical. Circuit privacy, on the other hand, requires that these distributions are the same as those of fresh ciphertexts.[1]

3.2 Classifications

Not all homomorphic schemes have the same properties. This part of the paper examines definitions that allow us to classify and distinguish between different types of schemes, depending on what circuits they can evaluate.

Definition 7 (Somewhat Homomorphic)**.** A $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) that has correct decryption and correct evaluation is called a *somewhat homomorphic encryption* scheme (SHE).

There is no requirement for compactness, so the ciphertexts can increase substantially in length with each homomorphic operation. Also, the set $\mathcal{C}$ of permitted circuits consists of *some* circuits; there is no requirement here as to which circuits this must include.

Definition 8 (Levelled Homomorphic [16, Def. 3.6])**.** A $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is called a *levelled homomorphic scheme* if it takes an auxiliary input $\alpha = d$ to Gen which specifies the maximum depth of circuits that can be evaluated. Further requirements are correctness, compactness and that the *length of the evaluation output* does not depend on d.

Other than circuit depth, there is no restriction on $\mathcal{C}$. If we require that $\mathcal{C}$ is the set of all binary circuits with depth at most d, the scheme is called *levelled fully homomorphic*.

The difference between somewhat and levelled homomorphic schemes is a potential point of confusion. The depth of circuits which a somewhat homomorphic encryption can handle can be increased through parameter choice – this usually means that the ciphertext size will increase with the depth of the circuits allowed. For a levelled homomorphic encryption scheme, the maximum depth is an input parameter and the length of the ciphertext does not depend on it.

Remark 3. The parameter α was introduced in Definition 1 specifically to allow specifying the maximum depth of circuit that can be evaluated. Thus, when we later assume that α is polynomial in λ, this is justified because in all existing schemes $\alpha = d$, a constant. However, we aim to work with the most general framework possible, so we also allow cases where α might have a different functionality and be substantially larger.

Definition 9 (Fully Homomorphic [16, Def. 3.5])**.** A *fully homomorphic encryption scheme* is a $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) that is compact, correct and where $\mathcal{C}$ is the set of all circuits.

[1] We would like to thank Shai Halevi for bringing this issue to our attention.

This definition means that the scheme can evaluate any circuit of arbitrary size, which does not need to be known when setting the parameters.

3.3 Evaluating in Stages

Sometimes we want to compute a result in two or more stages, where the results from one stage could be used as input for a later stage. In this case, we want to evaluate on ciphertexts that were output by Eval in addition to ciphertexts that were output by Enc.

The definition of correct evaluation (Definition 3) only guarantees that the algorithm Eval works when its input ciphertexts are in $\mathcal{X}$, the set of *fresh ciphertexts* that can be output by the Enc algorithm. We want to study under which conditions we can hope that the evaluation algorithm will work when given evaluation outputs (we present implications in Section 4.2).

Evaluation in stages is known as *i-hop homomorphic encryption* ([56, Section 2.2], [34, Section 1.4]), where i is either an integer or can be replaced by "multi","poly" or ∞ (see Definitions 12, 13 and 14 below). We now define *computation in stages* (also *staged computation*).

A computation $\mathbf{C}_{i,n}$ in i stages of width n is defined by a set of circuits $\{C_{k\ell}\}$ indexed by $1 \leq k \leq i, 1 \leq \ell \leq n$, where $C_{k\ell}$ has kn inputs. Given initial plaintexts $m_{01}, m_{02}, \ldots, m_{0n}$, we compute

$$m_{k\ell} = C_{k\ell}(m_{01}, m_{02}, \ldots, m_{0n}, \ldots, m_{k-1,1}, \ldots, m_{k-1,n})$$

for $1 \leq k \leq i$ and $1 \leq \ell \leq n$. The output of the staged computation after Eval and Dec is $m_{i1}, m_{i2}, \ldots, m_{in}$. Denoting the initial plaintexts by $\vec{m}_0$ and the output plaintexts by $\vec{m}_i$, we introduce the natural notation $\vec{m}_i = \mathbf{C}_{i,n}(\vec{m}_0)$.

Let (pk, evk, sk) be a key triple output by Gen, and let $c_{01}, c_{02}, \ldots, c_{0n}$ be a sequence of ciphertexts from $\mathcal{X}$. Compute the ciphertexts $\{c_{k\ell}\}$ for $1 \leq k \leq i, 1 \leq \ell \leq n$ recursively by

$$c_{k\ell} = \mathsf{Eval}(evk, C_{k\ell}, c_{01}, \ldots, c_{0n}, \ldots, c_{k-1,1}, \ldots, c_{k-1,n}).$$

The output of the encrypted staged computation is the sequence of ciphertexts $c_{i1}, c_{i2}, \ldots, c_{in}$. Denoting the fresh ciphertexts by $\vec{c}_0$ and the ouput ciphertexts by $\vec{c}_i$, we introduce the natural notation of Eval having multiple outputs

$$\vec{c}_i = \mathsf{Eval}(evk, \mathbf{C}_{i,n}, \vec{c}_0).$$

Remark 4. A slightly narrower view [34, 56] of computation in stages is that the only ciphertext output by one stage can be input for the next stage. Since it is normally considered possible to apply the identity function to a ciphertext, this formulation is usually no weaker than our more general view.

Let $\vec{c} = (c_1, \ldots, c_n)$ be a tuple of ciphertexts and $\vec{m} = (m_1, \ldots, m_n)$ be a tuple of plaintexts such that under a secret key sk, $\mathsf{Dec}(sk, c_k) = m_k$ for $1 \leq k \leq n$. We then introduce the natural notation

$$\vec{m} = \mathsf{Dec}(sk, \vec{c}).$$

Definition 10 (i-Hop Correctness)**.** Let pk, evk, sk be keys output by $\mathsf{Gen}(1^\lambda)$, and let $\mathbf{C}_{i,n} = \{C_{k\ell}\}$ be any staged computation where n is polynomial in λ and $\vec{c}_0 = (c_{01}, \ldots, c_{0n})$ in $\mathcal{X}^n$. A $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is *i-hop correct* if

$$\Pr[\mathsf{Dec}(sk, \mathsf{Eval}(evk, \mathbf{C}_{i,n}, \vec{c}_0) = \mathbf{C}_{i,n}(\mathsf{Dec}(sk, \vec{c}_0))] = 1 - \varepsilon(\lambda),$$

where ε is a negligible function and the probability is taken over the coins of the Eval algorithm invocations.

While previous definitions of i-hop (and multi-hop, see below) implicitly use a construction like i-hop correctness, it was never clearly defined in the literature. Additionally, we allow Eval to fail, although only with negligible probability.

In Figure 2 staged evaluation is illustrated for $i = 2$ and $n = 2$, where each invocation of Eval outputs n results, but not all of them must be used in the next iteration of Eval which is indicated by a dotted arrow. Furthermore, Eval may use $i \cdot n = 4$ different circuits in the whole process.

Now we have defined i-hop correctness, we can define i-hop, multi-hop, poly-hop and ∞-hop. Similar definitions of i-hop and multi-hop can be found in the work of Gentry et al. [34, Section 1.4] and Rothblum [56]. The main difference is that we allow inputs from each of the predecessor Eval algorithms as well as fresh ciphertexts. The previous definitions allow only the output of the direct predecessor Eval invocations as input.

Definition 11 (i-Hop, [34, 56])**.** Let $i \in \mathbb{N}$. We say that a $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is i-hop if j-hop correctness holds for all j with $1 \leq j \leq i$.

Remark 5 (FHE and i-Hop)*.* The relation between fully homomorphic and i-hop is another possible source of confusion. One may expect that if it is possible to evaluate an arbitrary circuit (fully homomorphic encryption), it would be possible to execute arbitrarily many circuits consecutively. This, however, is not the case. Outputs of Eval might look very different from *fresh* ciphertexts and there is no guarantee that they form valid inputs to Eval. For example, assume we have a 1-hop fully homomorphic encryption scheme, a circuit C that takes as input $c_1, \ldots, c_n$ and outputs $c'_1, \ldots, c'_v$, and a circuit C' that takes as input $c_1, \ldots, c_v$ and outputs $c'_1, \ldots, c'_w$. If we run $\mathsf{Eval}(evk, C' \circ C, c_1, \ldots, c_n)$ (where $C' \circ C$ is the concatenation of the two circuits), this is certainly a valid operation, because we are evaluating *one* circuit (not to be confused with staged computation). However, if we first run $\mathsf{Eval}(evk, C, c_1, \ldots, c_n)$ to obtain $c'_1, \ldots, c'_v$, then attempting to run $\mathsf{Eval}(evk, C', c'_1, \ldots, c'_v)$, is not supported by the 1-hop scheme, because $c'_1, \ldots, c'_v$ will not be valid inputs. This observation is important for applications where two separate entities compute on some encrypted data, and the second entity evaluates the output of the first. In this scenario, the second entity does not have access to the fresh ciphertexts and is forced to operate on the output of the evaluation given by the first. This would be impossible with a 1-hop scheme.

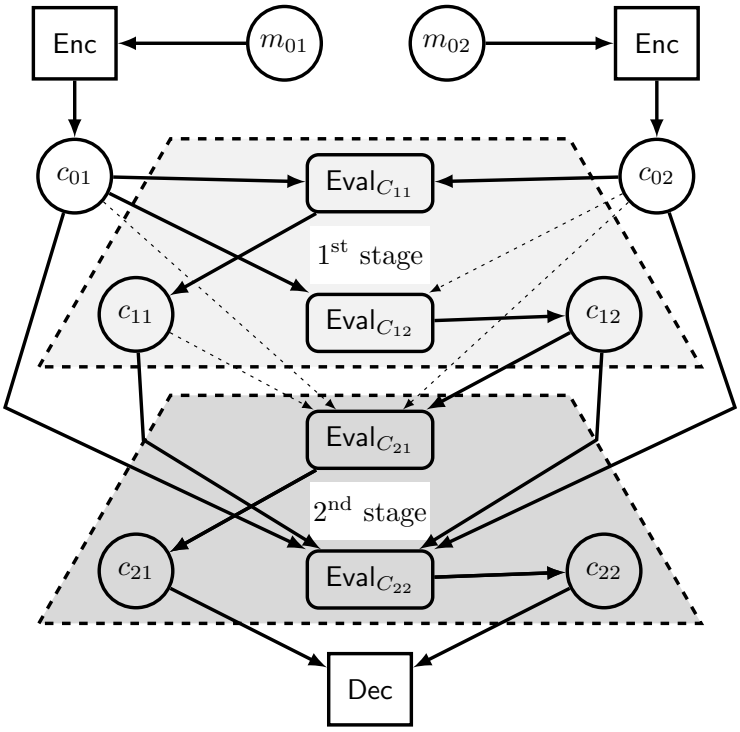

Figure 2: Diagram of staged evaluation for $i = 2$ stages and $n = 2$ in- and outputs. After encrypting the plaintexts, a subset of the resulting (fresh) ciphertexts is used for the following Eval algorithms as input successively. Therefore, this is the appropriate diagram for the formula $\vec{c}_2 = (c_{21}, c_{22}) \leftarrow \mathsf{Eval}(evk, \mathbf{C}_{2,2}, \vec{c}_0) = \mathsf{Eval}\left(evk, \mathbf{C}_{2,2}, (\mathsf{Enc}(pk, m_{01}), \mathsf{Enc}(pk, m_{02}))\right)$. Since a circuit does not have to use all given inputs, the dotted arrows represent the ignored inputs. Furthermore, the light shaded shapes belong to 1-hop and the darker shaded shapes to 2-hop.

Instead of being bounded by an integer, the hops may be bounded by some polynomial depending on λ which leads us to the next definition.

Definition 12 (Multi-Hop, [34, Sec. 1.4], [56])**.** Let p be some polynomial. We say that a $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is *multi-hop* if j-hop correctness holds for all j with $1 \leq j \leq p(\lambda)$.

Definition 13 (Poly-Hop)**.** Let p be some polynomial and let $\alpha \in \mathcal{A}$. We say that a $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is *poly-hop* if j-hop correctness holds for all j with $1 \leq j \leq p(\lambda, \alpha)$.

As far as we are aware, this is the first proposed definition of poly-hop. It seems to be a natural extension to the existing definitions of i-hop and multi-hop.

This way, the auxiliary input may influence the number of keys and therefore the number of possible evaluations.

Definition 14 (∞-Hop)**.** We say that a $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is ∞-*hop* if j-hop correctness holds for all j.

Again, as far as we know, ∞-hop was not yet mentioned in the literature. Like poly-hop, ∞-hop is a natural extension of the existing definitions. It allows an unlimited number of hops. Hence, there are direct implications for FHE. See Section 4 for further discussions on this topic.

Remark 6 (Hops and classifications)*.* We are not requiring a fully homomorphic scheme for the notion of hop correctness – the definition is applicable to any homomorphic scheme of Section 3.2 (somewhat homomorphic, levelled homomorphic, levelled fully homomorphic and fully homomorphic).

Remark 7 (Poly-hop vs. multi-hop)*.* What is the difference between poly-hop and multi-hop? If you need a security parameter to output a public key, then any bound on the security parameter is also a bound on the public key. This makes sense if a user cannot increase the size of the public key independently (or to some degree of independence) of the security parameter. This, however, is allowed by Definition 1, as some form of auxiliary input to the key generation algorithm. There is nothing stopping an auxiliary input defining the public key size, independent of the security parameter. This relates to poly-hop because in practice, levelled homomorphic schemes can be achieved by having several key pairs with which one can perform the recrypt operation (see Section 5.1 for a more detailed explanation). This means the public key size increases multiplicatively by this number of keys. Of course, if the number of key pairs is polynomial in λ (or more generally, if α is polynomial in λ), poly-hop and multi-hop are the same.

4 Implications

We now detail the implications of the definitions given in the previous section. First we return to the issue of compactness and its two, seemingly separate, definitions.

4.1 Consolidating compactness

As noted, there is a difference between Definition 4 and the definition of compactness originally given by Gentry [28]. This section is devoted to reconciliation of these two definitions of compactness. The definition presented by Gentry is given below, and also the definition of compact evaluation, which is important for Lemma 1.

Remark 8. Here, many results only hold if the auxiliary input to the key generation algorithm, α, is polynomially bounded by λ. For all meaningful applications (and for all homomorphic encryption schemes known to date) this appears to

be the case, but we cannot formally guarantee it (see also Remark 7). Thus, we state explicitly when we need this requirement for a statement to hold.

Definition 15 (G-Compactness [28, Def. 2.1.2])**.** A $\mathcal{C}$–evaluation scheme is *G-compact* if there is a polynomial f such that, for every value of the security parameter λ, the decryption algorithm can be expressed as a circuit C_D of size at most $f(\lambda)$.

Definition 16 (G-Compact Evaluation [28, Def. 2.1.3])**.** A $\mathcal{C}$–evaluation scheme (Gen, Enc, Eval, Dec) is said to *G-compactly evaluate* all permitted circuits in $\mathcal{C}$ if the scheme is G-compact and is correct for all permitted circuits.

Recall that the size of a circuit is just the total number of gates it has. Picturing a circuit as a directed graph, this is the sum of all the vertices minus the sum of the input vertices [44, §1.2, p.13]. It is not immediately clear that this definition of compactness is the same as the definition we gave earlier.

Theorem 1. *Let α be bounded by a polynomial in λ. A $\mathcal{C}$–evaluation scheme (*Gen*,* Enc*,* Eval*,* Dec*) G-compactly evaluates $\mathcal{C}$ if and only if the scheme compactly evaluates $\mathcal{C}$.*

The proof can be found in Appendix A.

Theorem 2. *A $\mathcal{C}$–evaluation scheme (*Gen*,* Enc*,* Eval*,* Dec*) with perfect circuit privacy implies compactness when α is polynomially bounded in λ.*

The proof can be found in Appendix B.

4.2 FHE and Hop Results

We now present results relating to FHE schemes and hop correctness, assuming that α is polynomially bounded by λ. Figure 3 shows a comprehensive overview of these results, also including Theorem 2. The diagram is formulated as in Figure 1, where there are two different kind of arrows. The simple black arrow is a requirement, so for example Theorem 4 has the two requirements of perfect circuit privacy and fully homomorphic. All of the requirements are needed for each theorem. The second arrow type is double-lined, which represents the implication of the theorem. As for Theorem 4, the given requirements yield an infinity-hop scheme.

Theorem 3. *A fully homomorphic encryption scheme (*Gen*,* Enc*,* Eval*,* Dec*) that is statistically circuit private is multi-hop.*

The proof can be found in Appendix C.

We now investigate the relationship between fully homomorphic and i-hop, noting what properties a somewhat homomorphic encryption scheme needs to be be *fully*. First, we examine under which conditions a fully homomorphic scheme allows infinite stages of computation (∞-hop):

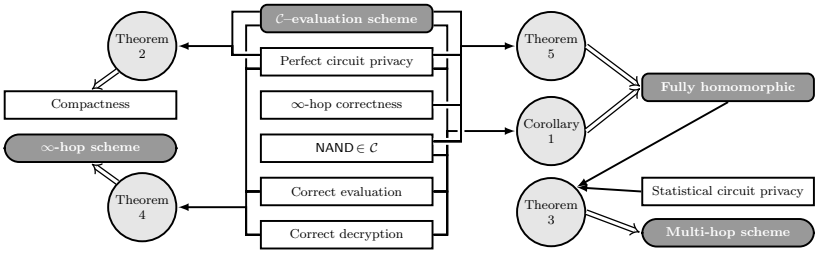

Figure 3: Overview of Theorem 2 through Theorem 5 and Corollary 1 and their dependencies. White rectangles are definitions, gray rounded rectangles are classifications, black shapes are hop schemes and light gray circles are theorems. Simple arrows pointing towards a theorem or corollary represent the requirements for a theoreom/corollary while double arrows represent the implication.

Theorem 4. *A somewhat homomorphic encryption scheme (*Gen*,* Enc*,* Eval*,* Dec*) which is perfect circuit private is ∞-hop.*

Proof. Since the scheme has perfect circuit privacy, the outputs of Eval are distributed identically to fresh encryptions. This means that they are of exactly the same form ($\mathcal{X} = \mathcal{Y} = \mathcal{Z}$) and decrypt *correctly*. So they are ciphertexts and constitute a valid input to Eval again. This holds no matter how often we apply evaluate, as the output is always of the same form as the input. □

The following theorem considers the other direction – when an ∞-hop scheme is fully homomorphic.

Theorem 5. *A somewhat homomorphic encryption scheme (*Gen*,* Enc*,* Eval*,* Dec*) with NAND in $\mathcal{C}$ that is perfect circuit private and ∞-hop is fully homomorphic.*

Proof. Since the scheme has perfect circuit privacy, it has compactness by Theorem 2. Thus, all we need to show is that the scheme can evaluate any circuit. Assume that this is not the case, so there exists a circuit C which the scheme cannot correctly evaluate. But then we can express C as a circuit composed only of NAND-gates. Since the scheme is ∞-hop and NAND $\in \mathcal{C}$, we can correctly evaluate each NAND-gate on the corresponding input, no matter what level of evaluation iteration this input has. Thus, we have found a way to correctly evaluate this circuit with the scheme, meaning $C \in \mathcal{C}$. This is a contradiction to our assumption and thus shows that the scheme is fully homomorphic. □

Corollary 1. *A somewhat homomorphic encryption scheme (*Gen*,* Enc*,* Eval*,* Dec*) that is perfect circuit private and has NAND $\in \mathcal{C}$ is fully homomorphic.*

Proof. By Theorem 2 and Theorem 4, then Theorem 5. □

5 Existing schemes

In this section we briefly survey existing fully homomorphic encryption schemes. Only limited steps towards full homomorphism were made before Gentry's breakthrough. Fellows and Koblitz's Polly Cracker [24] is fully homomorphic except that it lacks compactness. It was anyway not intended to be practical. Albrecht et al. [2] show that nearly all SHE schemes are variants of Polly Cracker. The Boneh-Goh-Nissim scheme [11] is compact, but can only handle a single multiplication. Obviously these schemes are not fully homomorphic by Definition 9, or by contemporary treatments of FHE [28].

Table 1 lists various prominent fully homomorphic encryption schemes, starting with Gentry's 2009 scheme. For each scheme the table mentions the underlying computational assumption (described below) and an indication of the asymptotic or concrete runtime where available.

5.1 Bootstrapping and Alternatives

A key concept in the development of the first fully homomorphic scheme is Gentry's *bootstrapping* technique. Schemes based on Gentry's blueprint are noise-based, which means that the plaintext is hidden by noise which can be removed by decryption. However, this noise increases with each homomorphic evaluation, and once it exceeds a certain threshold, decryption will fail.

To overcome this problem, Gentry introduced the notion of *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. As long as the evaluation algorithm can handle the decryption process plus one more gate, progress can be made in evaluating the circuit of interest.

Definition 17 (Bootstrappable)**.** A $\mathcal{C}$–evaluation scheme is called *bootstrappable* if it is able to homomorphically evaluate its own decryption circuit plus one additional NAND gate.

This informal definition essentially captures the more precise one in the literature [28]. Now the question is: Does publishing an encryption of the secret key under its own public key impair security?

If we assume it is safe to publish the encryption of the secret key under its corresponding public key, we achieve fully homomorphic encryption and even i-hop [28, 62, 17, 58]. This assumption is called *circular security*. However, if circular security does not hold then one possibility is to use a chain of public key/secret key pairs, where the secret key is always encrypted under the next public key. This allows suitable somewhat homomorphic schemes to become levelled homomorphic, where the level depends on the number of key pairs.

An alternative way to achieve homomorphic encryption is due to Brakerski et al. [15]. The challenge is still how to manage the noise, but this time it is achieved by reducing the modulus of the ciphertext space along with the noise.

Scheme	**Underlying Problems**	**Asymptotic Runtime**	**Concrete Runtime**
Gentry: A Fully Homomorphic Encryption Scheme [28]	BDDP & SSSP	$\mathcal{O}(\lambda^{3.5})$ per gate for ciphertext refreshing [60]	-
van Dijk, Gentry, Halevi, Vaikuntanathan: FHE over the Integers [62]	AGCD & SSSP	Public key size: $\mathcal{O}(\lambda^{10})$, no gate cost given	-
Coron, Naccache, Tibouchi: Public Key Compression and Modulus Switching for FHE over the Integers [20]	DAGCD & SSSP	Public key size: $\mathcal{O}(\lambda^5 \log(\lambda))$, no gate cost given	Recryption takes about 11 minutes.
Brakerski, Vaikuntanathan: Efficient FHE from (standard) LWE [16]	DLWE	Evaluation key size: $\mathcal{O}(\lambda^{2C} \log(\lambda))$ where C is a very large parameter that ensures bootstrappability.	-
Brakerski, Vaikuntanathan: FHE from Ring-LWE and Security for Key Dependent Messages [17]	PLWE	Very cheap key generation, unknown for bootstrapping	-
Brakerski, Gentry, Vaikuntanathan: FHE without Bootstrapping [15]	RLWE	Per-gate computation overhead $\mathcal{O}(\log \lambda \cdot \lambda \cdot d^3)$ (where d is the depth of the circuit) without bootstrapping, $\mathcal{O}(\log \lambda \cdot \lambda^2)$ with bootstrapping.	36 hours for an AES encryption on a supercomputer [32]. An updated implementation [33] runs AES-128 encryption with 2 seconds/block (3GB RAM). With bootstrapping 6 seconds/block 3.7GB RAM). In [40]: Vectors of 1024 elements from GF(2^{16}) were recrypted in 5.5 minutes at security level ≈ 76, single CPU core.
Smart, Vercauteren: FHE with Relatively Small Key and Ciphertext Sizes [58]	PCP & SSSP	Key generation is $\mathcal{O}(\log n \cdot n^{2.5})$ where n is the dimension of the lattice, according to [31]	Key generation took several hours even for small parameters which do not deliver a fully homomorphic scheme, for larger parameters the keys could not be generated.
Rohloff, Cousins: A Scalable Implementation of Fully Homomorphic Encryption Built on NTRU [55]	SVP & RLWE	-	Recryption at 275 seconds on 20 cores with 64-bit security.
Gentry, Halevi: Implementing Gentry's Fully-Homomorphic Encryption Scheme [31]	SVP & BDD	Key generation is $\mathcal{O}(\log n \cdot n^{1.5})$ where n is the dimension of the lattice	Bootstrapping: From 30 s for small setting, to 30 min for large setting.

Table 1: Selected fully homomorphic schemes and their underlying security problems. The authors often provide different runtime analyses for their schemes, so the figures may not be comparable. The concrete experiments have been run by the respective authors on widely different hardware, but still give an indication. Blank cells are, to the best of our knowledge, not publicly known.

A security parameter that dictates how small the modulus can be gives a bound on the number of levels. This line of work yields native levelled homomorphic schemes [16, 15, 55]. However, authors usually note that one can apply bootstrapping as an optimisation, as well as a means to get to a fully homomorphic i-hop scheme, again assuming circular security.

5.2 Security Assumptions

We now give a brief overview of the problems that existing schemes are based on. The formal definitions are often taken directly from the corresponding papers, but simplified by omitting parameters whenever possible. Many of these problems were studied by Ajtai [1].

Most of the problems below have reductions to either the *Shortest Vector Problem* (SVP) or the *Closest Vector Problem* (CVP), which informally requires a player to provide a shortest possible vector in the lattice and the vector closest to a point respectively. These problems have decisional variants as well. For instance, $GapSVP_\gamma$ is the problem of proving that there is a vector shorter than 1, or that all vectors are longer than γ. In addition, we mention the *shortest independent vector problem* (SIVP), which is essentially to compute a lattice basis with only short vectors.

We first consider the Learning With Errors problem family. All of these problems exist in both search and decision variants, just like the computational Diffie-Hellman and decisional Diffie-Hellman problems.

LWE: [53] The *Learning With Errors problem* is a generalization of the "learning parity with noise" problem.

For an integer $q = q(n)$ and an error distibution $\chi = \chi(n)$ on $\mathbb{Z}_q$, define the distribution $A_{s,\chi}$ for some $s \in \mathbb{Z}_q^n$ as the distribution obtained by choosing a vector $a \leftarrow \mathbb{Z}_q^n$ uniformly at random and a noise term $e \leftarrow \chi$ and outputting $(a, \langle a, s\rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. Then the (n, m, q, χ)-LWE problem is to output s, given m independent samples from $A_{s,\chi}$.

The decisional version is to distinguish between m samples chosen according to $A_{s,\chi}$ for some uniformly random s and m samples from the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

PLWE: [16] The *Polynomial LWE problem* is a variant of the Ring Learning With Errors Problem (RLWE) and is closely related to DLWE.

For a parameter λ, let $f(x) \in \mathbb{Z}[x]$ be a polynomial of degree $n = n(\lambda)$, and let $q = q(\lambda) \in \mathbb{Z}$ be a prime. Consider the rings $R = \mathbb{Z}[x]/\langle f(x)\rangle$ and $R_q = R/qR$, and let χ denote the Gaussian distribution over R. Then the $\mathrm{PLWE}_{f,q,\chi}$ assumption states that for all λ and for all $l = \mathrm{poly}(\lambda)$, the two distributions $D_1 = \{(a_i, a_i \cdot s + e_i)\}$ and $D_2 = \{(a_i, u_i)\}, i = 1, \ldots, l$, are computationally indistinguishable. Here, s, a_i and u_i are uniform in R_q and the e_i are sampled from χ.

RLWE: [49] This is the same problem as PLWE where $f(x) = x^d + 1$ and $d = d(\lambda)$ is a power of 2.

There also exists a variant with augmented data in the error term, named *Augmented LWE*. For certain parameters, A-LWE is as hard as LWE [8].

There exists a quantum reduction from LWE to SVP and SIVP by Regev [53]. It is also known that the search and decision variants are equally hard [47].

SSSP: [28] This is called the *Sparse Subset Sum Problem*. In his original work, Gentry "squashed the decryption circuit", which reduces the size of the decryption circuit such that it is in the set of circuits that the scheme can homomorphically evaluate. Idea: the secret key is written as the sum of some elements, and these elements are "hidden" in a much larger set of elements. This large set becomes part of the public key, and the secret key includes an indicator vector of which elements belong to the smaller set (i.e., sum up to the secret key). This gives the adversary information about the secret key, so we must ensure that it cannot be extracted from the public key. SSSP formalizes this requirement. Since all papers that follow Gentry's blueprint use this squashing technique, the SSSP problem appears several times in the table. It is formally defined as follows.

Let S and T be two natural numbers with $S \ll T$, and let q be a prime number. The challenger sets $b \leftarrow \{0, 1\}$. If $b = 0$, it generates a set τ with cardinality $|\tau| = T$ of uniformly random integers in $[-q/2, q/2]$ such that there exists a subset of cardinality S whose elements sum to $0 \bmod q$. If $b = 1$, the set τ is generated without this requirement. The challenge is to guess b.

BDD: The *Bounded Distance Decoding problem* is identical to the *Closest Vector Problem*, except that for BDD, there is a guarantee that the vector t is very close to the lattice. The Closest Vector Problem is a problem from lattice theory that informally asks for the point on a lattice that is closest to a given vector $t \in \mathbb{R}^n$.

There exists quantum reductions proving that GapSVP, BDD and SIVP are equally hard, as well as an equivalence between SVP and CVP [53]. An equivalence between SVIP and BDD, however, remains an open problem [48].

AGCD: [41] The *Approximate Greatest Common Divisor problem* is the task of given near multiples of a number p, to find that number p. Given polynomially many numbers of the form $x_i = q_i \cdot p + r_i$ where r_i is much smaller than $q_i \cdot p$, output p.

The decisional variant includes an an additional integer $z = x + b \cdot \alpha$. Here, x is of the same form as the x_i's, b is either 0 or 1, and α is from an appropriate interval depending on the parameters. The task is to find b.

PCP: The *Polynomial Coset Problem* [58] is a decisional problem that can informally be described as having to decide whether a given value is the evaluation of a small polynomial $\bmod p$, or randomly sampled from $\mathbb{F}_p$. More formally, we can describe the problem in a challenge scenario:

The challenger first selects $b \leftarrow \{0, 1\}$ randomly and runs the key generation algorithm of the scheme, which outputs a prime p and a value $\alpha \in \mathbb{F}_p$, derived under some constraints which we will not go into here. If $b = 0$, the challenger randomly chooses a polynomial $R(x)$ with coefficients in a certain range and computes $r = R(\alpha) \bmod p$. If $b = 1$, the challenger chooses $r \leftarrow \mathbb{F}_p$ randomly. The problem now is: Given (p, α, r), decide whether $b = 0$ or $b = 1$.

Notably, this problem differs from other problems in that it is defined with respect to a corresponding scheme, making it less natural and harder to ex-

plaining in outline. The PCP problem is related to the Ideal Coset Problem as defined by Gentry [28].

5.3 Implementations

It is fair to say that FHE mostly exists on paper. However, there also exist implementations, as suggested in the above table. The foremost among those is Halevi and Shoup's `HElib` [40], which implements the BGV scheme [15] along with optimisations such as ciphertext packing [59], which allows several plaintexts to be encoded in a single ciphertext. In 2014, bootstrapping was introduced to the library [40]. However, at the time of writing the secure Gaussian randomness distribution is not yet implemented, hence there are no security proofs for `HElib`.

The library was used by the IBM, Microsoft and Stanford/MIT teams at the 2015 iDASH Secure Genome Analysis Contest [45].

There exists another library called FHEW [22] which is based on the FHE scheme of Ducas and Micciancio [23].

6 Conclusion

In this paper we have simplified and structured the jungle of definitions in the field of homomorphic encryption. We investigated whether existing applications need homomorphic encryption as a solution to their problems, both in theory and in practice. Furthermore, we reviewed the current state of the art and presented it systematically.

There is still much work to be done. Current schemes have some way to go to be practical in daily applications. Thus we can expect continuing focus on making existing schemes more efficient and on constructing new efficient schemes. In fact, given that several applications do not require *fully* homomorphic encryption, an important and promising line of research is to identify applications which would benefit from appropriate homomorphic encryption schemes and afterwards tailor schemes for respective use cases.

Then there is the more theoretical line of work. While a framework for *group* homomorphic encryption schemes has been presented [5] and to some extent for FHE [4], an equivalent result is lacking for fully (or at least somewhat) homomorphic encryption schemes in full generality. As explained in Section 5, all secure schemes which go beyond simple group-homomorphic operations are noise-based and one of the main challenges is to control the noise. In fact this is often the reason why fully homomorphic encryption schemes are considerably less efficient. A unified view on somewhat/fully homomorphic encryption schemes may be very useful in gaining a better understanding of the expected security and on the possible design space.

All in all, the topic of FHE is an interesting and challenging research area with great potential, and there is much to be done. However, if research (specif-

ically the advancement of efficiency) continues at its current pace, we are confident that real-world applications may be right around the corner.

Acknowledgements

This work was supported by the German Academic Exchange Service (DAAD), project number 57068907 and the Juniorprofessoren-Programm Baden-Württemberg 2013.

References

[1] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 99–108. ACM, 1996.

[2] Martin R. Albrecht, Jean-Charles Faugère, Pooya Farshim, Gottfried Herold, and Ludovic Perret. Polly cracker, revisited. *Designs, Codes and Cryptography*, pages 1–42, 2015.

[3] Joël Alwen, Manuel Barbosa, Pooya Farshim, Rosario Gennaro, S. Dov Gordon, Stefano Tessaro, and David A. Wilson. On the relationship between functional encryption, obfuscation, and fully homomorphic encryption. In Martijn Stam, editor, *Cryptography and Coding – 14th IMA International Conference, IMACC 2013*, volume 8308 of *Lecture Notes in Computer Science*, pages 65–84. Springer, 2013.

[4] Frederik Armknecht, Stefan Katzenbeisser, and Andreas Peter. Shift-type homomorphic encryption and its application to fully homomorphic encryption. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology – AFRICACRYPT 2012*, volume 7374 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2012.

[5] Frederik Armknecht, Stefan Katzenbeisser, and Andreas Peter. Group homomorphic encryption: characterizations, impossibility results, and applications. *Des. Codes Cryptography*, 67(2):209–232, 2013.

[6] Frederik Armknecht and Thorsten Strufe. An efficient distributed privacy-preserving recommendation system. In *The 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop, Med-Hoc-Net 2011*, pages 65–70. IEEE, 2011.

[7] Sanjeev Arora and Boaz Barak. Computational complexity: A modern approach (draft). Accessed 27 Oct 2014, 2007.

[8] Rachid El Bansarkhani, Özgür Dagdelen, and Johannes A. Buchmann. Augmented learning with errors: The untapped potential of the error term. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography*

and Data Security, FC 2015, volume 8975 of *Lecture Notes in Computer Science*, pages 333–352. Springer, 2015.

[9] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[10] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Paterson [51], pages 149–168.

[11] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Ishai [42], pages 253–273.

[13] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.

[14] Christoph Bösch, Andreas Peter, Pieter H. Hartel, and Willem Jonker. SOFIR: securely outsourced forensic image recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014*, pages 2694–2698. IEEE, 2014.

[15] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.

[16] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 97–106. IEEE, 2011.

[17] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.

[18] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In Rabin [52], pages 483–501.

[19] Michael Clear, Arthur Hughes, and Hitesh Tewari. Homomorphic encryption with access policies: Characterization and new constructions. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on*

Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings, volume 7918 of *Lecture Notes in Computer Science*, pages 61–87. Springer, 2013.

[20] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 446–464. Springer, 2012.

[21] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini and Canetti [57], pages 643–662.

[22] Léo Ducas and Daniele Micciancio. Fhew: Fastest homomorphic encryption in the west. a fully homomorphic encryption library. `https://github.com/lducas/FHEW`. Accessed 2016-01-18.

[23] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015.

[24] Michael Fellows and Neal Koblitz. Combinatorial cryptosystems galore! In *Finite fields: theory, applications, and algorithms*, volume 168 of *Contemp. Math.*, pages 51–61. Amer. Math. Soc., Providence, RI, 1994.

[25] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 40–49. IEEE Computer Society, 2013.

[26] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Rabin [52], pages 465–482.

[27] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013*, volume 8270 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2013.

[28] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[29] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 169–178. ACM, 2009.

[30] Craig Gentry. Computing on the edge of chaos: Structure and randomness in encrypted computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:106, 2014.

[31] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In Paterson [51], pages 129–148.

[32] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Safavi-Naini and Canetti [57], pages 850–867.

[33] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. *IACR Cryptology ePrint Archive*, 2012:99, 2012.

[34] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable Yao circuits. In Rabin [52], pages 155–172.

[35] Henri Gilbert, editor. *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010.

[36] Oded Goldreich. Introduction to complexity theory, online lecture notes. Accessed 27 Oct 2014, 1999.

[37] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553. Springer, 2013.

[38] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2007.

[39] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 469–477. ACM, 2015.

[40] Shai Halevi and Victor Shoup. Bootstrapping for helib. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670. Springer, 2015.

[41] Nick Howgrave-Graham. Approximate integer common divisors. In Joseph H. Silverman, editor, *Cryptography and Lattices, International Conference, CaLC 2001*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001.

[42] Yuval Ishai, editor. *Theory of Cryptography – 8th Theory of Cryptography Conference, TCC*, volume 6597 of *Lecture Notes in Computer Science*. Springer, 2011.

[43] Arjan Jeckmans, Andreas Peter, and Pieter H. Hartel. Efficient privacy-enhanced familiarity-based recommender system. In Jason Crampton et al., editors, *Computer Security – ESORICS 2013*, volume 8134 of *Lecture Notes in Computer Science*, pages 400–417. Springer, 2013.

[44] Stasys Jukna. *Boolean Function Complexity – Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012.

[45] Kristin Lauter. Practical applications of homomorphic encryption, 2015.

[46] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 1219–1234. ACM, 2012.

[47] Vadim Lyubashevsky. Search to decision reduction for the learning with errors over rings problem. In *2011 IEEE Information Theory Workshop, ITW 2011*, pages 410–414. IEEE, 2011.

[48] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2009.

[49] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Gilbert [35], pages 1–23.

[50] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW*, pages 113–124. ACM, 2011.

[51] Kenneth G. Paterson, editor. *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*. Springer, 2011.

[52] Tal Rabin, editor. *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.

[53] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM, 2005.

[54] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[55] Kurt Rohloff and David Bruce Cousins. A scalable implementation of fully homomorphic encryption built on NTRU. In Rainer Böhme et al., editors, *Financial Cryptography and Data Security – FC 2014 Workshops, BITCOIN and WAHC 2014*, volume 8438 of *Lecture Notes in Computer Science*, pages 221–234. Springer, 2014.

[56] Ron Rothblum. Homomorphic encryption: From private-key to public-key. In Ishai [42], pages 219–234.

[57] Reihaneh Safavi-Naini and Ran Canetti, editors. *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*. Springer, 2012.

[58] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[59] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.

[60] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2010.

[61] Top Threats Working Group. The notorious nine: Cloud computing top threats in 2013. Report, Cloud Security Alliance, February 2013. Accessed 7 Apr 2015.

[62] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Gilbert [35], pages 24–43.

[63] Marten van Dijk and Ari Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In Wietse Venema, editor, *5th USENIX Workshop on Hot Topics in Security, HotSec'10*. USENIX Association, 2010.

[64] Zhiqiang Yang et al. Privacy-preserving classification of customer data without loss of accuracy. In Hillol Kargupta et al., editors, *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, pages 92–102. SIAM, 2005.

A Proof of Theorem 1

We prove Theorem 1 with the following lemmas.

Lemma 1. *A scheme with G-compactness and correctness is compact.*

This lemma holds only when we assume correct decryption and correct evaluation. To understand why we must have correctness, consider G-compactness without it. In this case, we have a decryption circuit of polynomial size but it does not have to *actually* decrypt. This means we could form a trivial decryption circuit that takes any Eval output, trims the bits to a specified maximum size and runs the decryption circuit on that. Obviously, this would not bound the output of Eval, so G-compactness without correctness does not imply compactness.

Proof. Given the security parameter λ, we can find a decryption circuit with size at most $p(\lambda)$. As each gate takes at most 2 inputs, the bound on the number of inputs to the circuit is less or equal to $2p(\lambda) = q(\lambda)$ where q is a polynomial.[2] This bound on the input length means that the output from the Eval algorithm must output a ciphertext less than $q(\lambda)$ bits in length. Otherwise, we would be unable to run the decryption algorithm *correctly*, contradicting our assumption. Noting that $q(\lambda)$ is a bound independent of the size of the circuit being evaluated, this gives us compactness. □

Lemma 2. *A scheme with compactness and correctness is G-compact when α is polynomially bounded by λ.*

Proving this lemma relies on results from complexity theory, allowing us to construct a polynomially sized circuit from a polynomial algorithm. For details on the relationship between algorithm running time and circuit size see Gentry [30, §2.1 Circuits], Goldreich [36, Ch. 2, Ch. 20 §1.2] or Arora & Barak [7, Ch. 6].

Proof. All outputs of Eval are no more than $b(\lambda)$ bits long, meaning input to the decryption algorithm after evaluation is at most $b(\lambda)$ bits in length. Eval is a poly-time algorithm, and so its running time is bounded polynomially by the length of its input.

The algorithm Enc also has a polynomial bound on the length of its output. Firstly note, the number of outputs from an algorithm cannot be greater than the algorithm running time. Next, we know Enc is a poly-time algorithm taking two inputs, pk and $m \in \mathcal{P}$, where pk is an output from Gen, a poly-time algorithm taking the input parameters λ and $\alpha = \alpha(\lambda)$. $\mathcal{P}$ is described in pk. Now, the running time of Enc is bounded by some polynomial on the input parameters, themselves bounded polynomially by the input parameter λ. Thus, the output from Enc is bounded by a polynomial, $a(\lambda)$ say. Since sk is again an output of Gen, we can also bound its length by a polynomial $c(\lambda)$. Taking $d(\lambda) = \max\{a(\lambda), b(\lambda)\}$, which is also a polynomial in λ, we can define $v = d + c$. Clearly v is a polynomial in λ and bounds the size of the inputs to Dec. Thus, Dec has a running time of $p(v(\lambda))$ for some polynomial p.

[2] Any circuit where each gate takes n inputs (for some bounded n) can be constructed as a circuit with gates taking at maximum two inputs, with only a constant factor increase in size. [36, §1.2]

Using the results cited above, we can now construct a decryption circuit that replicates the algorithm, where the circuit size will be some polynomial q on the running time of the algorithm. Thus the size of the decryption circuit is $q(p(v(\lambda)))$, which is a polynomial, independent of $\mathcal{C}$. This completes the proof. □

B Proof of Theorem 2

We prove Theorem 2 with the following lemmas.

Lemma 3. *A $\mathcal{C}$–evaluation scheme (*Gen*,* Enc*,* Eval*,* Dec*) with perfect circuit privacy implies $\mathcal{X} = \mathcal{Y}$.*

Lemma 4. *A $\mathcal{C}$–evaluation scheme (*Gen*,* Enc*,* Eval*,* Dec*) with $\mathcal{X} = \mathcal{Y}$ implies compactness when α is polynomially bounded in λ.*

Recall that $\mathcal{X}$ is the set of all possible outputs of the encryption algorithm and $\mathcal{Y}$ is the set of all possible outputs of the evaluation algorithm.

When we are dealing with perfect circuit privacy, perfect indistinguishability means that

$$|\Pr[\mathsf{Enc}(pk, C(\mathsf{Dec}(sk, c_1), \ldots, \mathsf{Dec}(sk, c_n))) = x] - \Pr[\mathsf{Eval}(evk, C, c_1, \ldots, c_n) = x]| = 0$$

holds for every key tuple (pk, sk, evk) output by $\mathsf{Gen}(1^\lambda, \alpha)$, for all $c_i \in \mathcal{X}$ and all $C \in \mathcal{C}$.

Proof of Lemma 3. $\mathcal{Y} \subseteq \mathcal{X}$:

Assume $\mathcal{Y} \nsubseteq \mathcal{X}$, then there exists $a \in \mathcal{Y}$ such that $a \notin \mathcal{X}$. Hence a is a possible output of Eval with probability $p > 0$, but is not a possible output of Enc. This means that for some $c_i \in \mathcal{X}$, and some $C \in \mathcal{C}$

$$|\Pr[\mathsf{Enc}(pk, C(\mathsf{Dec}(sk, c_1), \ldots, \mathsf{Dec}(sk, c_n))) = a] - \Pr[\mathsf{Eval}(evk, C, c_1, \ldots, c_n) = a]| = |0 - p| = p > 0,$$

which is a contradiction to perfect circuit privacy.

For $\mathcal{X} \subseteq \mathcal{Y}$, we use the strategy from above. So $\mathcal{X} \subseteq \mathcal{Y}$ and $\mathcal{Y} \subseteq \mathcal{X}$, hence $\mathcal{X} = \mathcal{Y}$. □

Proof of Lemma 4. If $\mathcal{X} = \mathcal{Y}$ then all outputs of Eval are also outputs of Enc. Thus, by the argument in Lemma 1 on the length of outputs of Enc,

$$\text{length}(c) \leq a(\lambda)$$

for some polynomial a for all fresh ciphertexts c, where λ is the input parameter. Hence, all evaluation outputs are bounded by some polynomial on the input parameter. □

C Proof of Theorem 3

Proof. We first show that the probability that the evaluation algorithm outputs a ciphertext that is not fresh or does not decrypt correctly is negligible. The result then follows from the structure of a computation in stages.

Since the scheme is fully homomorphic, there is a negligible function ε' such that if pk, evk, sk have been output by $\mathsf{Gen}(1^\lambda, \alpha)$, then for any circuit $C \in \mathcal{C}$ and ciphertexts $c_1, c_2, \ldots, c_n \in \mathcal{X}$, we have

$$\Pr[\mathsf{Dec}(sk, \mathsf{Eval}(evk, C, c_1, \ldots, c_n)) = C(\mathsf{Dec}(sk, c_1), \ldots, \mathsf{Dec}(sk, c_n))] = 1 - \varepsilon'(\lambda).$$

Since the scheme has statistical circuit privacy, there is a negligible function ε'' such that if pk, evk, sk have been output by $\mathsf{Gen}(1^\lambda, \alpha)$, then for any circuit $C \in \mathcal{C}$ and ciphertexts $c_1, c_2, \ldots, c_n \in \mathcal{X}$, we have

$$\Delta = \sum_{y \in \mathcal{Z}} |\Pr[\mathsf{Enc}(pk, C(\mathsf{Dec}(sk, c_1), \ldots, \mathsf{Dec}(sk, c_n))) = y] - \Pr[\mathsf{Eval}(evk, C, c_1, \ldots, c_n) = y]| \leq \varepsilon''(\lambda).$$

Since Enc will never output ciphertexts in $\mathcal{Z} \setminus \mathcal{X}$, we have

$$\begin{aligned} &\Pr[\mathsf{Eval}(evk, C, c_1, \ldots, c_n) \in \mathcal{X}] \\ &\geq 1 - \sum_{y \in \mathcal{Z} \setminus \mathcal{X}} \Pr[\mathsf{Eval}(evk, C, c_1, \ldots, c_n) = y] \\ &\geq 1 - \Delta \geq 1 - \varepsilon''(\lambda). \end{aligned}$$

It is then clear that there is a negligible function ε such that the probability for an evaluation to be correct and the resulting ciphertext is in $\mathcal{X}$ is at least $1 - \varepsilon(\lambda)$.

Suppose pk, evk, sk have been output by $\mathsf{Gen}(1^\lambda, \alpha)$, that $\mathbf{C}_{i,n}$ is a computation in i stages of width n and that $c_{01}, c_{02}, \ldots, c_{0n} \in \mathcal{X}$. We are interested in the probability

$$\Pr[\mathsf{Dec}(sk, \mathsf{Eval}(evk, \mathbf{C}_{i,n}, \vec{c}_0)) = \mathbf{C}_{i,n}(\mathsf{Dec}(sk, \vec{c}_0))].$$

Let E_j be the event that after the jth stage, all of the ciphertexts computed so far are in $\mathcal{X}$ and decrypt to the correct value. It is clear that $\Pr[E_i]$ is no greater than the probability we are interested in. Since different executions of Eval are independent, we have:

$$\Pr[E_j] \geq \Pr[E_j \mid E_{j-1}] \Pr[E_{j-1}] \geq (1 - \epsilon(\lambda))^n \Pr[E_{j-1}].$$

It quickly follows that

$$\Pr[E_i] \geq (1 - \epsilon(\lambda))^{in}.$$

To conclude the proof, we must show that $1 - (1 - \epsilon(\lambda))^{in}$ is negligible when in is polynomial in λ. For simplicity, we will write ϵ instead of $\epsilon(\lambda)$ and show that $1 - (1 - \epsilon)^k$ is negligible when k is polynomial in λ.

We will show this by induction:

1. $k = 1 : 1 - (1 - \epsilon)^1 = \epsilon$, which is negligible by definition.

2. Now let $\mu := 1 - (1 - \epsilon)^k$ be negligible. Then we have:
   $1 - (1 - \epsilon)^{k+1} = 1 - (1 - \epsilon)^{k+1} - \mu + \mu$
   $= 1 - (1 - \epsilon)^{k+1} - (1 - (1 - \epsilon)^k) + \mu$
   $= (1 - \epsilon)^k - (1 - \epsilon)^{k+1} + \mu$
   $= (1 - \epsilon)^k \cdot (1 - (1 - \epsilon)) + \mu$
   $= (1 - \epsilon)^k \cdot (-\epsilon) + \mu$
   This shows that by increasing the exponent from k to $k + 1$, we get an increase of $(1 - \epsilon)^k \cdot (-\epsilon)$. If we can show that this increase is negligible, we have completed our proof. We again show this by induction:

   (a) $k = 1 : (1 - \epsilon) \cdot (-\epsilon) = \epsilon^2 - \epsilon$. Noting that $|\epsilon^2 - \epsilon| < \epsilon$ and ϵ negligible, the claim holds for $k = 1$.

   (b) Let $\alpha := (1 - \epsilon)^k \cdot (-\epsilon)$ be negligible. Then we have:
   $(1 - \epsilon)^{k+1} \cdot (-\epsilon) = (1 - \epsilon) \cdot (1 - \epsilon)^k \cdot (-\epsilon)$
   $= (1 - \epsilon) \cdot \alpha < \alpha$,
   so for the case of $k + 1$ it is also negligible.

3. Thus, we have shown that we start out with something negligible and add something negligible in each step. So, as long as we take polynomially many steps, the result will always also be negligible.

□