Computing on the edge of chaos: Structure and randomness in encrypted computation

Craig Gentry

Abstract. This survey, aimed mainly at mathematicians rather than practitioners, covers recent developments in homomorphic encryption (computing on encrypted data) and program obfuscation (generating encrypted but functional programs). Current schemes for encrypted computation all use essentially the same "noisy" approach: they encrypt via a noisy encoding of the message, they decrypt using an "approximate" ring homomorphism, and in between they employ techniques to carefully control the noise as computations are performed. This noisy approach uses a delicate balance between structure and randomness: structure that allows correct computation despite the randomness of the encryption, and randomness that maintains privacy against the adversary despite the structure. While the noisy approach "works", we need new techniques and insights, both to improve efficiency and to better understand encrypted computation conceptually.

Mathematics Subject Classification (2010). Primary 68Qxx; Secondary 68P25.

Keywords. Cryptography, complexity theory, homomorphic encryption, software obfuscation, learning with errors (LWE).

1. Introduction

Many results in cryptography are counterintuitive. Alice and Bob can agree on a secret key over a public channel. Alice can prove to Bob that she knows something – say, a proof that P $\neq$ NP – without revealing any details of the proof. Alice can send Bob an encryption of her data $m_1, \ldots, m_t$ such that Bob can compute a succinct encryption of $f(m_1, \ldots, m_t)$ for any function f that he wants, but without Bob learning anything about $m_1, \ldots, m_t$. The last trick is called "fully homomorphic encryption" (FHE). This survey is about FHE and another type of encrypted computation called program obfuscation. Obfuscation allows Alice to encrypt a software program so that the obfuscated program is fully executable but hides essential secrets inside.

Before exploring encrypted computation, let us review some basics about computation and cryptography, illustrated by the story of a young theoretical computer scientist.

1.1. Computation. Young Gauss, the story goes, was challenged by his teacher to add up the numbers from 1 to 100. To his teacher's surprise, Gauss computed the solution almost instantly, while the other pupils toiled for the remainder of the class.

While his classmates added the numbers sequentially, Gauss found a shortcut. He saw that, for even n, the first n numbers can be partitioned into $n/2$ pairs that each add up to

Proceedings of the International Congress of Mathematicians, Seoul, 2014

$n+1$, and that therefore the sum of the first n numbers is $n(n+1)/2$. A mathematician might say that Gauss found a formula or expression for the sum of the first n numbers – namely, $n(n+1)/2$. A computer scientist would add that Gauss also found an *algorithm* or *program*. Moreover, Gauss's algorithm is *efficient*, in contrast to the *inefficient* algorithm used by his classmates.

Gauss's algorithm for adding up the first n numbers takes as input the number n, represented by $k = \log_2 n$ bits (or $\log_{10} n$ decimal digits). The most complex part of Gauss's algorithm is to multiply n and $n+1$, which requires $O(k^2)$ steps using grade-school multiplication. Since the number of computational steps in Gauss's algorithm is only polynomial in the size of the input, we say his algorithm is *polynomial-time*. The sequential algorithm used by his classmates takes at least $n = 2^k$ steps, which is *exponential-time*.

If a problem – such as adding up the numbers 1 to n, or multiplying two numbers – has a polynomial-time algorithm that always solves it, then we say the problem is in the complexity class P (for polynomial-time). BPP, which contains P, is the class of problems solvable by efficient algorithms, which includes probabilistic polynomial-time (PPT) algorithms that may use random coins and only solve the problem with good probability. NP (for "nondeterministic polynomial-time") contains problems that, if you happen to guess the solution, you can verify that it is correct in polynomial time. For example, the integer factorization problem – decomposing an integer N into its prime factors, which is essentially the inverse of multiplication – is in NP, but widely believed not to be in BPP. The biggest open problem in complexity theory is to prove $\mathsf{P} \neq \mathsf{NP}$ (if that is the case).

1.2. Cryptography. Since we have not resolved $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ and other complexity-theoretic questions, we do not know whether strong cryptography is possible. We might live in any of Impagliazzo's Worlds [23]. Impagliazzo imagined a face-off between Gauss and his teacher in five different worlds, each of which is possible given what we currently know. In "Algorithmica", $\mathsf{P} = \mathsf{NP}$ or some moral equivalent, making much of modern cryptography insecure, and making it virtually impossible for the teacher to stump Gauss. To make the face-off fair, the teacher's problem needs to have a succinct verifiable answer, but any such problem is in NP, hence in P, and therefore is easy for Gauss to solve. At the other extreme, in "Cryptomania", public-key cryptography [12, 33] is possible: two parties can communicate secret messages over public channels. Impagliazzo notes "In Cryptomania, Gauss is utterly humiliated. By means of conversations in class, [the teacher] and his pet student would be able to jointly choose a problem that they would both know the answer to, but which Gauss could not solve." Most cryptographers bet their careers that we live in Cryptomania. But betting against the Gausses of the world is a risky proposition, and so "cryptographers seldom sleep well" [25].

Still, cryptographers soldier on. An early triumph was a paper by Goldwasser and Micali [21] that introduced "probabilistic encryption", defined a rigorous (now standard) notion of security for encryption schemes, and proposed an elegant construction of public-key encryption whose security they provably reduced to a natural, plausible computational assumption: that the quadratic residuosity problem is hard. We review their results here as a vigorous warm-up for recent encrypted computation schemes.

A public-key encryption scheme has three efficient algorithms: a key-generation algorithm K that generates public and secret keys $(\mathrm{pk}, \mathrm{sk})$, an encryption algorithm E that takes pk and a plaintext message m and outputs a ciphertext c, and a decryption algorithm D that takes sk and c and recovers m. It is called "public key", since anyone can use the publicly

available pk to encrypt (without needing any secret knowledge). Of course, for any key pair $(\mathrm{pk}, \mathrm{sk})$ output by K, whenever $c = \mathsf{E}(\mathrm{pk}, m)$, it should hold that $m = \mathsf{D}(\mathrm{sk}, c)$.

Goldwasser and Micali observed that, to be secure, an encryption scheme really should be *probabilistic* – that is, E needs to be randomized, and there must be many ciphertexts for each plaintext. If E were deterministic, an adversary could easily detect whether two ciphertexts encrypt the same thing! To make this intuition more precise, they defined a notion of "semantic security" for encryption in terms of a game between a challenger and an adversary. In the initial phase, the adversary can ask the challenger for encryptions of messages of its choosing. (In the public-key setting, the adversary can generate these encryptions itself.) Then, the adversary generates two equal-length messages m_0, m_1 and asks for an encryption of one of them. The challenger sends a "challenge ciphertext" $\mathsf{E}(m_b)$ for random $b \in \{0, 1\}$, the adversary wins the game if it guesses b, and the scheme is considered semantically secure if the adversary has negligible advantage.

In the Goldwasser-Micali (GM) public-key encryption scheme, Alice samples random prime integers p, q according to an appropriate distribution and sets $N = pq$, samples a uniform $x \in (\mathbb{Z}/N\mathbb{Z})^*$ that is a non-square modulo N but whose Jacobi symbol $\left(\frac{x}{N}\right)$ equals 1, and publishes (N, x) as her public key. Bob encrypts $m \in \{0, 1\}$ for Alice by sampling random $r \in (\mathbb{Z}/N\mathbb{Z})^*$ and sending the ciphertext $c \leftarrow x^m \cdot r^2 \in (\mathbb{Z}/N\mathbb{Z})^*$. That is, an encryption of 0 is a square, and an encryption of 1 is a non-square (with Jacobi symbol 1). Alice decrypts to recover m by distinguishing whether c is a square modulo the secret prime factor p (e.g., by using Gauss's quadratic reciprocity theorem).

The quadratic residuosity problem is related to the integer factorization problem. The problem is: given a composite integer $N = pq$ (but not the prime factors p and q) and an element $x \in (\mathbb{Z}/N\mathbb{Z})^*$ whose Jacobi symbol is 1 (where N and x are sampled according to appropriate distributions), decide whether x is a square in $(\mathbb{Z}/N\mathbb{Z})^*$. The quadratic residuosity assumption is that the quadratic residuosity problem is hard (not in BPP). To put it another way, the assumption is that, against all PPT adversaries, the subset of squares modulo $N = pq$ is pseudorandom among the set of elements with Jacobi symbol 1. The assumption is clearly stronger than factoring, but it seems like a safe assumption, since we do not know an actual algorithm to solve it that is significantly faster than factoring. For us, the assumption has the added appeal of taunting our adversary Gauss, since he can use his quadratic reciprocity theorem to compute the Jabobi symbol of x modulo N without knowing N's factorization, but this does not help him since we always fix $\left(\frac{x}{N}\right) = 1$.

To reduce the semantic security of their scheme to quadratic residuosity, Goldwasser and Micali use a "hybrid argument" approach that has become standard. Assume that our adversary Gauss can break the cryptosystem – i.e., can distinguish encryptions of 0 from encryptions of 1. Consider two different games, Game 0 and Game 1. In Game 0, we generate the public key (N, x) and a challenge ciphertext (an encryption of a random bit $m \in \{0, 1\}$) for Gauss in the correct way. By assumption, Gauss should be able to guess m with noticeable advantage. In Game 1, however, we generate the public key (N, x) in a different way. Specifically, we make x a square in $(\mathbb{Z}/N\mathbb{Z})^*$, and generate the challenge ciphertext by encrypting m using the normal encryption procedure, as if (N, x) were a valid public key. In Game 1, encryptions of 0 and encryptions of 1 have the same distribution (either way, the ciphertext is a random square), and thus Gauss cannot have any advantage guessing m. Thus, Gauss's success probability noticeably differs in Games 0 and 1. To construct a PPT algorithm to decide whether x is a non-square or square (i.e., whether we are in Game 0 or Game 1), we simply use Gauss's performance to help us distinguish. This

bases the security of GM on quadratic residuosity.

1.3. Homomorphic encryption. The GM scheme has a curious bonus feature: it is *malleable*. It allows anyone to manipulate (in limited but meaningful ways) what is encrypted, even without knowing the secret key: to *compute on encrypted data*. Specifically, suppose that c_1 is a GM encryption of $m_1 \in \{0,1\}$, and c_2 is a GM encryption of m_2 – that is, $c_1 = x^{m_1} \cdot r_1^2$ and $c_2 = x^{m_2} \cdot r_2^2$ for some $r_1, r_2 \in (\mathbb{Z}/N\mathbb{Z})^*$. We can increment the plaintext by multiplying the ciphertext by x, without even knowing what the plaintext is. The new ciphertext $c \leftarrow c_1 \cdot x = x^{m_1+1} \cdot r_1^2$ encrypts $m_1 + 1$. Also, we can add plaintexts by multiplying the corresponding ciphertexts: $c \leftarrow c_1 \cdot c_2 = x^{m_1+m_2} \cdot (r_1 r_2)^2$ encrypts $m_1 + m_2$. These plaintext additions are in $\mathbb{Z}/2\mathbb{Z}$, since x^2 is an encryption of 0. Interestingly, GM allows an unlimited number of plaintext additions, but GM's overall malleability is limited. GM can compute linear functions on encrypted data, but it does not (for example) provide any way to operate on two ciphertexts so as to multiply the two plaintexts.

Rivest, Adleman and Dertouzos [32] saw the potential of computing on encrypted data a few years earlier in 1978, shortly after the invention of the RSA public-key encryption scheme [33], which allows multiplications of plaintexts but not additions. They wondered whether it could be possible to construct an encryption scheme that is *completely* malleable, that allows *unlimited* computations on encrypted data. They called such a scheme a "privacy homomorphism". These days, we call it "fully homomorphic encryption" (FHE), where "fully" means it allows any computation over encrypted values. (GM is "additively homomorphic" and RSA is "multiplicatively homomorphic".) They also foresaw that an FHE scheme would have amazing applications. It took more than 30 years after Rivest et al. proposed the notion to discover the first plausible FHE scheme [16]. Now that we have discovered plausible constructions, we have made tremendous progress improving them, but still have far to go.

Before we address what FHE can do, let us be more precise about what it is. In this survey, an FHE scheme is first of all a public-key encryption scheme with the usual algorithms K, E, and D. Let $\mathcal{M}$ and $\mathcal{C}$ be the message space and ciphertext space of the scheme. Let us say that a ciphertext $c \in \mathcal{C}$ encrypts a message $m \in \mathcal{M}$ under key $(\mathrm{pk}, \mathrm{sk})$ if decryption returns $m \leftarrow \mathsf{D}(\mathrm{sk}, c)$. The special feature of an FHE scheme is that it comes equipped with a *fourth* efficient algorithm, called *Evaluate* and denoted by V, such that for any valid key pair $(\mathrm{pk}, \mathrm{sk})$, any t (for any t) encryptions $c_1, \ldots, c_t$ of any messages $m_1, \ldots, m_t \in \mathcal{M}$ under $(\mathrm{pk}, \mathrm{sk})$, and for any t-ary function $f : \mathcal{M}^t \to \mathcal{M}$, $\mathsf{V}(\mathrm{pk}, f, c_1, \ldots, c_t)$ outputs a ciphertext c that encrypts $f(m_1, \ldots, m_t)$. Crucially, Evaluate is a public algorithm that anyone can execute without the secret key, and of course we want the encryption scheme to be semantically secure despite its availability. In short, an FHE scheme allows computation of any function f *inside* an "impenetrable box" of encryption.

We can describe FHE in terms of a commutative diagram.

$$\begin{array}{ccc} \mathcal{C}^t & \xrightarrow{\mathsf{V}(\mathrm{pk}, f, \cdot, \ldots, \cdot)} & \mathcal{C} \\ \Big\downarrow{\scriptstyle \mathsf{D}(\mathrm{sk}, \cdot, \ldots, \cdot)} & & \Big\downarrow{\scriptstyle \mathsf{D}(\mathrm{sk}, \cdot)} \\ \mathcal{M}^t & \xrightarrow{f(\cdot, \ldots, \cdot)} & \mathcal{M} \end{array}$$

The diagram is meant to convey that, for any key, messages, ciphertexts, and function f, the order of decryption and applying f does not matter: either way we end up with

$f(m_1, \ldots, m_t)$. An analogous commutative diagram with encryption instead of decryption does not work. Although it is true that the order of encryption and applying f does not matter in the sense that (either way) we end up with an encryption of $f(m_1, \ldots, m_t)$, the actual ciphertexts might be different. (Recall that having many different ciphertexts for each message is essential for an encryption scheme to be semantically secure.)

Later in the survey, we will see in detail how to construct an FHE scheme. At this point, we must keep the reader in suspense.

1.3.1. Applications of homomorphic encryption. An exciting potential application of FHE is preserving privacy online, which is more relevant now than ever before. For example, we seem to heading toward widespread acceptance of cloud computing, where users put their data online "in the cloud" for convenience and availability. Putting everything online unencrypted is to risk an Orwellian future, not just because the corporation hosting our data may misuse it, but also because a government may strong-arm the corporation into providing a backdoor. For certain types of data, such as medical records, storing them off-site unencrypted may be illegal. On the other hand, encrypting one's data seems to nullify the benefits of the "computing" part of cloud computing. Unless I give the cloud my secret decryption key (sacrificing my privacy), how can I expect the cloud to do any meaningful processing of my encrypted data? Fully homomorphic encryption provides a way out of this false dilemma. If I want to make some query f on my encrypted data, I can just send a description of f to the cloud, which uses the Evaluate algorithm to derive an encryption of $f(m_1, \ldots, m_t)$, which is the response to my query.

In addition to encrypting my data, I can encrypt my query f (under the same pk). More broadly, I can encrypt a *program* P, so that the cloud can execute P on unencrypted data or data encrypted under the same pk, and output the encrypted result. At first, this fact may seem surprising, but it is just an application of Turing's idea that a program can be viewed just another type of data to be processed by a universal Turing machine. (In more modern terms, a program can be read and executed by an interpreter program.)

The applications of FHE may seem counterintuitive and hard to believe. In a world with FHE – call it "Cryptomegalomania" – cryptography flexes its muscles and sticks its tongue out at young Gauss. Gauss might have the last laugh though. Current FHE schemes are too impractical to realize all of the applications that are possible in principle. Developing a significantly faster FHE scheme is an interesting mathematical problem that also has high stakes for society.

1.3.2. Shortcomings of homomorphic encryption. Besides high overhead, there are two related "problems" with FHE.

The first problem is that Evaluate always has an *encrypted output*. This is, in some sense, optimal for security: nothing is ever revealed to anyone but the secret key holder. But it is often sub-optimal for functionality. Sometimes it is useful to reveal some (carefully controlled) *unencrypted* information to the Evaluator. This especially true for *encrypted programs*. One might like to hide (encrypt) certain aspects of a program (e.g., to prevent it from being semantically deconstructed) while preserving its functionality as a fully executable program with unencrypted inputs and outputs.

The second problem is that, while FHE can handle general computations "efficiently" in the sense of "polynomial-time", FHE cannot exploit certain optimizations essential to the practicality of computation in modern computing environments. Specifically, FHE needs to

put a function f or program P into a special format – called a boolean or arithmetic *circuit* – before it can be processed.[1] In a circuit evaluation of f, the number of computational steps does not depend on the input x. For the security of FHE, this is necessary: if the run time of Evaluate depended on the particular value of (encrypted) x, it would reveal something about x. However, it also means that Evaluate's run time depends on the *worst-case* x's; Evaluate can never take a shortcut for "easy" inputs. Similarly, FHE cannot do *random access* (as in a random access machine (RAM)) over encrypted data, since FHE does not allow the Evaluator to learn unencrypted data-dependent addresses. Nor does FHE allow an Evaluator to exploit an *inverted index*, which helps make searches (like web searches) over huge data-sets practical.

1.4. Program obfuscation. Using FHE, we can generate encrypted programs that have *encrypted output*. But is there some way to generate encrypted programs that have *unencrypted output*? To put it another way: Is there any meaningful sense in which we can "encrypt" a program while preserving its functionality (input/output behavior) as a fully executable program? This is the seemingly-paradoxical and hard-to-define goal of *program obfuscation*.

Program obfuscation may sound impossible to achieve, and indeed some notions of obfuscation are. For example, consider a program P that prints its own code. Since any obfuscation $O(P)$ of P must have the same functionality as P, $O(P)$ reveals P completely. Barak et al. [4] showed that some programs are unobfuscatable even without being so exhibitionist. They showed that, assuming one-way functions (functions that are hard to invert), there are *unlearnable* programs P (programs for which no PPT algorithm can recover P or any code equivalent to P just from oracle access to P) that can be completely recovered from any code that implements them. Obfuscation is impossible in an "absolute" sense: for some programs, any obfuscation reveals everything.

However, it turns out that obfuscation is possible in a "relative" sense. To understand this notion of encrypting a program, let us revisit what it means to encrypt a message. Goldwasser and Micali called an encryption scheme "semantically secure" if a PPT adversary has negligible advantage of winning the following game: the adversary picks two equal-length messages m_0, m_1, the challenger encrypts one of them, and the adversary tries to guess which one. They need the "equal-length" message requirement, because a ciphertext always reveals some information about the message it encrypts – namely, an upper bound on its length. Similarly, an obfuscated program always reveals something about the original program – an upper bound on its size, and also the program's input/output behavior. Accordingly, Barak et al. [4] defined an analogue of semantic security for programs via a similar game: the adversary picks two equal-size functionally-equivalent programs (represented as circuits C_0, C_1), the challenger obfuscates one of them, and the adversary tries to guess which one. The obfuscator is considered secure if every PPT adversary has negligible advantage of winning the game. This notion is called *indistinguishability obfuscation* (IO).

It is not obvious that IO is actually useful. An IO obfuscator does not guarantee it will hide any secrets residing in the program. It does not provide any absolute guarantees about the quality of the obfuscation. However, IO provides a strong relative guarantee – namely, an indistinguishability obfuscator is a "best-possible" obfuscator: it is as good as any other obfuscator of roughly the same complexity [4, 22]. To see this, suppose O is a secure indistinguishability obfuscator. Suppose $BO(\cdot)$ is the actual best obfuscator of a

[1] We will discuss circuits in more detail in Section 2.

certain complexity of circuits of a certain size, whereas $Pad(\cdot)$ merely increases the size of circuits the same amount as $BO(\cdot)$. Then, for any circuit C, the circuits $BO(C)$ and $Pad(C)$ are the same size and have the same functionality, and so $O(BO(C))$ and $O(Pad(C))$ are indistinguishable. Since they are indistinguishable, $O(Pad(C))$ obfuscates C as well as $O(BO(C))$, which obfuscates C as well as $BO(C)$.

Although IO only provides a relative guarantee of security, it can be used to construct schemes having absolute guarantees. For example, Garg et al. [15] showed how to use IO to construct a functional encryption scheme [5]: a public-key scheme administered by an authority that chooses a function f and distributes secret keys to users such that a user with sk_y associated to string y can recover exactly $f(x, y)$ from a ciphertext encrypting x. For example, y might specify a user's security clearance, and f might specify a redaction policy, such that user y obtains only the portion of document x for which it has clearance. Obfuscation can also be used to "fix" some of the problems with FHE. For example, it can be used to allow encrypted computation in the RAM model of computation (rather than circuits) [1, 18].

Garg et al. [15] recently found the first plausibly secure construction of IO. Here is a very brief overview of how their scheme works. First, they show how to "bootstrap" IO for NC^1 (logarithmic depth) circuits to IO for general circuits. Specifically, the obfuscation of a circuit C consists of encryptions of C under two FHE key pairs $(\mathrm{sk}_0, \mathrm{pk}_0)$, $(\mathrm{sk}_1, \mathrm{pk}_1)$ and an obfuscated conditional decryption circuit $O(\mathsf{ConD})$ (to be described momentarily). The Evaluator computes the encrypted program C on its input under both FHE public keys, and feeds the resulting ciphertexts, with a "proof" that they were computed correctly, as input to $O(\mathsf{ConD})$, which decrypts one ciphertext using sk_0 if the proof verifies. Garg et al. use the fact that ConD can be implemented in NC^1 for known FHE schemes. Assuming O is a secure IO for NC^1, they show that a PPT attacker cannot distinguish whether the FHE secret key inside $O(\mathsf{ConD})$ is sk_0 or sk_1, since either way ConD's output is the same. This shell game shows that sk_0 is hidden, and forms part of their hybrid security proof for IO for general circuits.

Next, Garg et al. present an indistinguishability obfuscator for NC^1 circuits. Their NC^1 obfuscator uses a graded encoding scheme by Garg et al. [14]. A graded encoding scheme is similar to a homomorphic encryption scheme, with the important difference that it comes equipped with zero test that allows anyone to efficiently distinguish when the encoded value is 0. This zero test allows some unencrypted information to leak (unlike FHE), but schemes using graded encodings are carefully designed to ensure that (hopefully) this leakage can only occur when the Evaluator computes over the encodings in a permitted way. Currently, known schemes for IO for NC^1 have security based on unconventional assumptions about graded encodings.

Since Garg et al.'s obfuscation construction, there have been some improvements both in security and efficiency, but both aspects are still worse than for FHE, in part because current obfuscation schemes use FHE as a *component*. This is a young and active area of research.

1.5. "Computing on the Edge of Chaos" and "Structure and Randomness". We now begin turning to the construction of FHE and obfuscation schemes. Before we begin in earnest, let us start with a high-level intuition for how current FHE schemes (and the obfuscation schemes derived from them) work. Current FHE schemes all use essentially the same "noisy" approach. They encrypt via a noisy encoding of the message: by sending the message to a ciphertext that is similar to a perturbed codeword in an error-correcting code. The

decrypter recovers the message by recovering the noise. The public key is, in some sense, a "bad" basis of the error-correcting code, which permits efficient encryption but does not permit efficient correction of errors. By careful manipulation of the ciphertexts, an Evaluator can add and multiply the underlying plaintexts while increasing the noise by only a small amount. Furthermore, when the noise becomes almost large enough to drown out the signal (the message), the Evaluator can apply an operation called "bootstrapping" to "refresh" the noisy ciphertext: to generate a new ciphertext that encrypts the same message but with less noise. In short, the "noise" turns out to be both a boon and a bane. The noise hides the message from adversaries. However the noise lies behind the impracticality of current schemes: it makes ciphertexts large and it requires computationally expensive steps to bound the noise as computations are performed.

The phrases "computing on the edge of chaos" and "structure and randomness" capture some intuitions that I have about encrypted computation, and the possibility that a noisy approach may be necessary. Of course, these intuitions may be illusions. I would like nothing more than for someone to find a radically different way of constructing fully homomorphic encryption and obfuscation schemes that escapes the current paradigm of using noisy, approximate homomorphisms. Consider the title of this paper a provocation, a challenge.

My (not very strongly held) intuition, for what it's worth, is that "exact" mathematical structures – e.g., exact rather than approximate homomorphisms of the kind used in previous weakly homomorphic encryption schemes such as Goldwasser-Micali – seem either too rigid (e.g., they allow only additive but not multiplicative homomorphism) or too permissive (e.g., they allow full homomorphism but enable trivial linear algebra attacks). Instead, for robust encrypted computation, we seem to need mathematical structures that can be inexact without simply being wrong – that is, structures that noisily remain close to exact solutions.

To be secure under Goldwasser and Micali's notion of "semantic security", an encryption scheme must be probabilistic – i.e., it must use randomness in encryption. But getting this randomness to play nicely with the structure we need for correct computation is a delicate balance, and it raises certain questions: What happens to the randomness when we do homomorphic operations on ciphertexts? Does the randomness mix with the structured part of the ciphertexts, or does it somehow remain cordoned off? If the former, how is the structure preserved (so as to allow correct decryption)? If the latter, how does the randomness remain safely cordoned off despite performing complex general computations? (It seems like general computation would induce a lot of mixing.) Also, in the latter case, how does the scheme remain secure – for example, how does it remain secure against linear algebra attacks if the randomness is perpetually isolated to certain coordinates? In the noisy approach to homomorphic encryption, the randomness indeed mixes with the structure (in particular, with the message), but the randomness is always kept small so that it does not overwhelm the structure.

I thought "computing on the edge of chaos" would be a fun and original way to describe the current approach to encrypted computation, but it turns out the phrase has already been taken. Apparently, it refers to a critical phase transition point in cellular automata between overly ordered and completely chaotic where the automata become capable of universal computation, and more broadly refers to the notion that dynamic "lifelike" systems, such as the economy or human brain, are healthiest when they are "poised on the edge of chaos". The notion seems intuitively appealing, though there has been pushback against it as being unrigorous and unsubstantiated. The idea that the noisy approach to encrypted computation somehow exploits a phase transition between order and chaos also seems intuitively

appealing, if even more unsubstantiated.

1.6. Roadmap. In the rest of the survey, we will limit our focus to FHE. We will describe in depth how to construct an FHE scheme with security provably based on the hardness of the so-called learning with errors (LWE) problem.

2. Circuits and homomorphic encryption

We touched upon circuits and homomorphic encryption in the Introduction. Here, we discuss them more formally.

2.1. Circuits. Before we can specify how to Evaluate a function using homomorphic encryption, we need to be more explicit about our *model of computation*. The canonical theoretical representation of a computer is the Turing machine, described by Alan Turing in the 1930's. It handles general computations, and is as efficient as modern random access memory (RAM) computers up to polynomial factors (assuming the RAM computer's memory is not pre-loaded). However, in this survey, we will primarily use a mathematically cleaner representation of algorithms, called a boolean or arithmetic *circuit*. Circuits also handle general computations, and almost as efficiently as Turing machines. In particular, if there is a Turing machine program that always evaluates a function f in at most T_f steps, then there is a circuit for f that has size $O(T_f \cdot \log T_f)$ [30].

An arithmetic circuit is a remarkably simple and mathematically clean way of representing a program. It is typically just a composition of addition gates (which take several inputs and output their sum), multiplication gates (which take several inputs and output their product), and scalar multiplication gates (which take one input and multiply it by a scalar), where these operations are performed over some ring. The gates are typically arranged into levels, so that the outputs of gates at level i are inputs to gates at level $i + 1$ unless i is the last level of the circuit. The circuit cannot contain any loops (it is a directed acyclic graph), but one can reuse the output of a gate as input to multiple higher-level gates. The number of gates is called the *size* of the circuit, and the number of levels is called the *depth*. Notice that, since the circuit just uses addition and multiplication, the output of each gate has a nice mathematical interpretation: it is simply a multivariate polynomial (evaluated at the inputs).

When the ring is $\mathbb{F}_2$ and each gate has at most two inputs, we call the circuit a boolean circuit. Interestingly, any boolean function can be computed using a circuit composed entirely of NAND gates. For $x, y \in \{0, 1\}$, $\text{NAND}(x, y) = 1 - x \cdot y \in \{0, 1\}$. Restricting to $\{0, 1\}$, we can implement NAND over any ring.

It may be surprising that multivariate polynomials representable by polynomial-size circuits, even boolean circuits of NAND gates, are adequate to represent polynomial-time computation.[2] However, a multivariate polynomial with low circuit complexity may be very complex by other measures. Even when the circuit has polynomial size, the multivariate polynomials it represents may have an exponential number of monomials. Moreover, over large fields, the degree of the polynomials may be exponential in the depth of the circuit,

[2]Leslie Lamport, in his essay *How to Tell a Program from an Automobile*, remarked that "An automobile runs, a program does not. (Computers run, but I'm not discussing them.) ... An automobile is a piece of machinery, a program is some kind of mathematical expression". Lamport's observation becomes especially clear when the program is represented as a circuit, which in turn represents nothing more than a set of multivariate polynomials.

since each level of multiplication gates may double the degree.

2.2. Homomorphic encryption. A homomorphic encryption scheme is a tuple of four probabilistic polynomial time (PPT) algorithms $(\mathsf{K}, \mathsf{E}, \mathsf{D}, \mathsf{V})$. In this survey, the message space $\mathcal{M}$ of the scheme will always be some ring and our computational model will be arithmetic circuits over this ring (e.g., addition, multiplication and NAND gates).

- $\mathsf{HE.K}$ takes the security parameter λ (and possibly other parameters of the scheme) and produces a secret key sk and a public key pk.
- $\mathsf{HE.E}$ takes pk a message $m \in \mathcal{M}$ and produces a ciphertext c which is the encryption of m.
- $\mathsf{HE.D}$ takes sk and a ciphertext c and produces a message m.
- $\mathsf{HE.V}$ takes pk, an arithmetic circuit f over $\mathcal{M}$, and ciphertexts $c_1, \ldots, c_t$, where t is the number of inputs to f, and outputs a ciphertext c.

Roughly speaking, the security parameter λ specifies the security level of the scheme. The algorithms of the scheme should take time $\text{poly}(\lambda)$, but any known algorithms to attack the scheme should take time super-polynomial in λ, preferably exponential (say 2^{λ}) time.

Definition 2.1 (Correctness and Compactness). We say that a homomorphic encryption scheme $(\mathsf{K}, \mathsf{E}, \mathsf{D}, \mathsf{V})$ *correctly evaluates* a circuit family $\mathcal{F}$ if for all $f \in \mathcal{F}$ and for all $m_1, \ldots, m_t \in \mathcal{M}$ it holds that if sk, pk were properly generated by K with security parameter λ, and if $c_i = \mathsf{E}(\text{pk}, m_i)$ for all i, and $c = \mathsf{V}(\text{pk}, f, c_1, \ldots, c_t)$, then

$$\Pr[\mathsf{D}(\text{sk}, c) \neq f(m_1, \ldots, m_t)] = \mathsf{negl}(\lambda) ,$$

where the probability is taken over all the randomness in the experiment.

We say that the scheme *compactly evaluates* the family if in addition the run time of the decryption circuit only depends on λ and not on its input.

The notation $\mathsf{negl}(\lambda)$ means the function grows more slowly than the inverse of any polynomial: $\mathsf{negl}(\lambda) = O(1/\lambda^c)$ for any constant c.

The reason for the compactness requirement is that homomorphic encryption is uninteresting without it. If the ciphertext size could depend on the circuit size, we could just set $c = (f, c_1, \ldots, c_t)$, and decrypt c by decrypting the c_i's and applying f. Obviously such a scheme is useless for delegation of computation, since the decrypter rather than the Evaluator performs all of the computation.

Much of this survey will focus on the construction of a *leveled* fully homomorphic scheme, where the parameters of the scheme depend (polynomially) on the depth (but not the size) of the circuits that the scheme is capable of evaluating.

Definition 2.2 (Leveled FHE). We say that a family of homomorphic encryption schemes $\{\mathcal{E}^{(L)} : L \in \mathbb{Z}^+\}$ is leveled fully homomorphic if, for all $L \in \mathbb{Z}^+$, they all use the same decryption circuit, $\mathcal{E}^{(L)}$ compactly evaluates all circuits of depth at most L, and the computational complexity of $\mathcal{E}^{(L)}$'s algorithms is polynomial (the same polynomial for all L) in the security parameter, L, and (in the case of the evaluation algorithm) the size of the circuit.

In a "pure" FHE scheme, the complexity of the algorithms (except for Evaluate) is independent of L.

We use Goldwasser and Micali's notion of semantic security [21].

Definition 2.3. A homomorphic scheme is secure if any PPT adversary that first gets a properly generated pk, then specifies $m_0, m_1 \in \mathcal{M}$ and finally gets $\mathsf{E}(\mathsf{pk}, m_b)$ for random b, cannot guess b with probability $> 1/2 + \mathsf{negl}(\lambda)$.

Of course, the adversary can try to use the additional Evaluate algorithm to win the semantic security game.

3. Learning with Errors (LWE)

As we saw in the Introduction, when cryptographers construct an encryption scheme, they try to *prove* that the scheme is secure as long as a natural problem (such as quadratic residuosity) is hard to solve. This proof is called a *reduction*. Here, we describe a natural problem called *learning with errors* (LWE). Later, we will show how to construct public-key and homomorphic encryption schemes whose security reduces to it. We also review some evidence that LWE is a hard problem.

The LWE problem was introduced by Regev [31]. Informally, the "search" version of LWE is about solving "noisy" systems of linear equations. The problem is to recover a n-dimensional vector $\vec{s}$ over $\mathbb{Z}/q\mathbb{Z}$ from many pairs $(\vec{a}_i, b_i)$, where the $\vec{a}_i$'s are sampled as uniformly random vectors over $\mathbb{Z}/q\mathbb{Z}$, and b_i is set to $\langle \vec{a}_i, \vec{s} \rangle + e_i \in \mathbb{Z}/q\mathbb{Z}$ for some "error" e_i of small magnitude ($\ll q$). If not for the errors, we could recover $\vec{s}$ efficiently using Gaussian elimination after receiving about n equations. Introducing error seems to make the problem hard.

More formally, LWE is typically defined as a "decision" problem as follows.

Definition 3.1 (LWE). For security parameter λ, let $n = n(\lambda)$ be an integer dimension, $q = q(\lambda) \geq 2$ be an integer, and $\chi = \chi(\lambda)$ be a distribution over $\mathbb{Z}$. The $\mathrm{LWE}_{n,q,\chi}$ problem is to distinguish the following two distributions:

(1) Output $(\vec{a}_i, b_i)$ sampled uniformly from $(\mathbb{Z}/q\mathbb{Z})^{n+1}$.

(2) For fixed uniform $\vec{s} \leftarrow (\mathbb{Z}/q\mathbb{Z})^n$, sample $\vec{a}_i \leftarrow (\mathbb{Z}/q\mathbb{Z})^n$ uniformly, sample $e_i \leftarrow \chi$, set $b_i = \langle \vec{a}_i, \vec{s} \rangle + e_i \in \mathbb{Z}/q\mathbb{Z}$, and output $(\vec{a}_i, b_i)$.

The $\mathrm{LWE}_{n,q,\chi}$ assumption is that the $\mathrm{LWE}_{n,q,\chi}$ problem is hard.

For $n, q = \mathrm{poly}(\lambda)$, Regev gave a polynomial-time reduction from search LWE to decision LWE. Applebaum et al. [2] showed that the hardness of LWE is unaffected when the coefficients of secret $\vec{s}$ are chosen from the small error distribution χ.

Sometimes we prefer to view LWE in the following way. Let $\vec{c}_i = (b_i, \vec{a}_i)$ and $\vec{t} = (1, -\vec{s})$ for $b_i, \vec{a}_i, \vec{s}$ as above. Then $[\langle \vec{c}_i, \vec{t} \rangle]_q = [\vec{e}_i]_q$ is small for all i, where $[x]_q$ denotes the representative of x in $(-q/2, q/2]$. The LWE problem is to decide whether there exists a vector $\vec{t}$ that is "nearly orthogonal" to all of the $\vec{c}_i$'s.

Typically, χ is taken to be a discrete Gaussian distribution over $\mathbb{Z}$, with deviation $\sigma \ll q$. Rather than referring explicitly to the noise distribution χ, sometimes it is convenient to refer to a bound β on the size of the noise.

Definition 3.2 (β-bounded distributions). A distribution ensemble $\{\chi_n\}_{n \in \mathbb{N}}$, supported over the integers, is called β-bounded if $\Pr_{e \leftarrow \chi_n}[|e| > \beta] = \mathsf{negl}(n)$.

When the noise is extremely small or has some structure, there are sub-exponential algorithms to solve LWE [3]. For example, when $e_i \in \{0, 1\}$ for all i, solving LWE is easy: taking tensor products, $\langle \vec{c}_i, \vec{t} \rangle \in \{0, 1\}$ implies $\langle \vec{c}_i \otimes \vec{c}_i, \vec{t} \otimes \vec{t} \rangle - \langle \vec{c}_i, \vec{t} \rangle = 0$, giving us a $O(n^2)$-dimension error-free linear system to recover $\vec{t} \otimes \vec{t}$, hence $\vec{t}$. However, for discrete Gaussian error distributions with $\sigma = \text{poly}(n)$, the hardness of LWE stops depending so much on the noise bound β, and appears to depend more on the ratio q/β.

In particular, the LWE problem has been shown to be as hard *on average* (for random instances) as certain lattice problems *in the worst-case* (the hardest instances). A n-dimensional lattice is a (full-rank) additive subgroup of $\mathbb{R}^n$. For lattice dimension parameter n and number d, the shortest vector problem GapSVP$_\gamma$ is the problem of distinguishing whether a n-dimensional lattice has a nonzero vector of Euclidean norm less than d or no nonzero vector shorter than $\gamma(n) \cdot d$. The gist of the theorem below is that if one can solve average-case n-dimensional LWE for ratio q/β then one can solve worst-case n-dimensional GAPSVP$_\gamma$ for γ just a little larger than q/β.

Theorem 3.3 ([26, 27, 29, 31], Corollary 2.1 from [6]). *Let $q = q(n) \in \mathbb{N}$ be either a prime power or a product of small (size* poly(n)*) distinct primes, and let $\beta \geq \omega(\log n) \cdot \sqrt{n}$. Then there exists an efficient sampleable β-bounded distribution χ such that if there is an efficient algorithm that solves the average-case LWE problem for parameters n, q, χ, then:*

- *There is an efficient quantum algorithm that solves GapSVP$_{\tilde{O}(nq/\beta)}$ on any n-dimensional lattice.*
- *There is an efficient classical algorithm that solves GapSVP$_{\tilde{O}(nq/\beta)}$ on any n-dimensional lattice when $q \geq \tilde{O}(2^{n/2})$.*

Brakerski et al. [9] recently improved the classical result by removing the requirement on the size of q.

GAPSVP$_\gamma$ is NP-hard for any constant γ, but unfortunately in cryptography we need γ to be larger (at least n in the theorem above). For $\gamma = \text{poly}(n)$, the fastest algorithm to solve GAPSVP$_\gamma$ takes time $2^{O(n)}$. (As a crude rule of thumb, the fastest algorithm to solve GAPSVP$_{2^k}$ takes roughly $2^{n/k}$ time [34].) Interestingly, there are no quantum algorithms for GAPSVP that perform significantly better than classical algorithms. In contrast, there are polynomial-time quantum algorithms for integer factorization and some other common problems used in cryptography.

4. Public key encryption from LWE

Regev [31] described a simple encryption scheme based on LWE. We describe a variant of his scheme here. We split key generation algorithm K into three parts Setup, $\mathsf{SecretKeyGen}$ and $\mathsf{PublicKeyGen}$. Let $[x]_q$ denote the integer $x \in (-q/2, q/2]$ that represents the coset of $x \in \mathbb{Z}/q\mathbb{Z}$.

- $\mathsf{Setup}(1^\lambda)$: Choose an odd integer modulus $q = q(\lambda)$, lattice dimension parameter $n = n(\lambda)$, and error distribution $\chi = \chi(\lambda)$ appropriately for LWE for security parameter λ. Also, choose parameter $m = m(\lambda) = O(n \log q)$. Let $params = (n, q, \chi, m)$.
- $\mathsf{SecretKeyGen}(params)$: Sample $\vec{s} \leftarrow \chi^n$. Set $\text{sk} = \vec{t} \leftarrow (1, -s_1, \ldots, -s_n) \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$.

- $\mathsf{PublicKeyGen}(params, \mathrm{sk})$: Generate a matrix $A \leftarrow (\mathbb{Z}/q\mathbb{Z})^{m \times n}$ uniformly and a vector $\vec{e} \leftarrow \chi^m$. Set $\vec{b} = A \cdot \vec{s} + \vec{e}$. Set B to be the $(n+1)$-column matrix consisting of $\vec{b}$ followed by the n columns of A. Set the public key $\mathrm{pk} = B$. (*Remark:* Observe that $B \cdot \vec{t} = \vec{e}$.)
- $\mathsf{E}(params, \mathrm{pk}, \mu)$: To encrypt message $\mu \in \{0,1\}$, sample uniform $\vec{r} \in \{0,1\}^m$, set $\vec{\mu} \leftarrow (\mu, 0, \ldots, 0) \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$, and output the ciphertext:

$$\vec{c} \leftarrow \vec{\mu} + 2 \cdot \vec{r} \cdot B \in (\mathbb{Z}/q\mathbb{Z})^{n+1}.$$

- $\mathsf{D}(params, \mathrm{sk}, \vec{c})$: Output $[[\langle \vec{c}, \vec{t} \rangle]_q]_2$.

Decryption works correctly when the parameters are set so that $|\langle \vec{r}, \vec{e} \rangle| < q/4 - 1$ is guaranteed, since if $\vec{c} = \vec{\mu} + 2 \cdot \vec{r} \cdot B$ for $\mu \in \{0,1\}$, then $[\langle \vec{c}, \vec{t} \rangle]_q = [\mu + 2 \cdot \langle \vec{r}, \vec{e} \rangle]_q$ is an integer of magnitude $< q/2$ with the same parity as μ.

Interestingly, the encryption process of Regev's scheme already uses the fact the scheme is additively homomorphic. Each row $2 \cdot B_i$ of $2 \cdot B$ is an encryption of 0, in the sense that $[\langle B_i, \vec{t} \rangle]_q$ is small and even. To encrypt, one takes a random subset sum (defined by $\vec{r}$) of the $2 \cdot B_i$'s to obtain a "random" encryption of 0, and then one adds in $\vec{\mu}$ to get an encryption of μ.

This encryption process increases the size of the error: the error associated to the ciphertext is μ plus a subset sum of the errors associated to the $2 \cdot B_i$'s. One needs to set q large enough to "accommodate" the error expansion – again, one wants $|\mu + 2 \cdot \langle \vec{r}, \vec{e} \rangle| < q/2$ to ensure correct decryption.

The security of Regev's scheme follows from the following lemma [31].

Lemma 4.1 (Implicit in [31]). *Let $params = (n, q, \chi, m)$ be such that the $\mathrm{LWE}_{n,q,\chi}$ assumption holds, with q odd. Then, for $m = O(n \log q)$ and B, $\vec{r}$ as generated above, the joint distribution $(B, 2 \cdot \vec{r} \cdot B)$ is computationally indistinguishable from uniform over $(\mathbb{Z}/q\mathbb{Z})^{m \times (n+1)} \times (\mathbb{Z}/q\mathbb{Z})^{n+1}$. Concretely, it suffices to take $m > 2n \log q$.*

The lemma says that, for Regev's encryption scheme, it is hard to distinguish a uniform matrix and uniform vector from a valid pk and a valid encryption of 0.

To sketch a proof of the lemma, observe that it follows from two claims: that it is hard to distinguish $(B, 2 \cdot \vec{r} \cdot B)$ from $(U, 2 \cdot \vec{r} \cdot U)$ where U is uniform in $(\mathbb{Z}/q\mathbb{Z})^{m \times (n+1)}$, and also $(U, 2 \cdot \vec{r} \cdot U)$ from $(U, \vec{u})$ where $\vec{u}$ is uniform in $(\mathbb{Z}/q\mathbb{Z})^{n+1}$. The first claim follows immediately from the LWE assumption, since given a LWE instance B or U, we can generate the $2 \cdot \vec{r} \cdot B$ or $2 \cdot \vec{r} \cdot U$ part ourselves. The second claim is true *statistically*. For large enough m, the distributions $(U, 2 \cdot \vec{r} \cdot U)$ and $(U, \vec{u})$ have negligible statistical distance from each other when q is odd.

Now, let us use the lemma to reduce LWE to the semantic security of Regev's encryption scheme. Assume an adversary wins the semantic security game with non-negligible advantage. We imagine two games between the challenger and the adversary. In Game 0, the challenger uses the distribution $(B, 2 \cdot \vec{r} \cdot B)$ to generate its public key $\mathrm{pk} = B$ and challenge ciphertext $\vec{c} \leftarrow \vec{\mu} + 2 \cdot \vec{r} \cdot B$. By assumption, the adversary guesses μ with non-negligible advantage. In Game 1, uses uniform $(U, \vec{u}) \in (\mathbb{Z}/q\mathbb{Z})^{m \times (n+1)} \times (\mathbb{Z}/q\mathbb{Z})^{n+1}$, sets $\mathrm{pk} = U$, and sets $\vec{c} \leftarrow \vec{\mu} + \vec{u}$. In Game 1, since $\vec{u}$ is uniform, the adversary has no advantage guessing μ. We guess that the distribution is $(B, 2 \cdot \vec{r} \cdot B)$ (that we are in Game 0) if the adversary guesses μ correctly; otherwise, we guess the distribution is uniform (that we are in Game 1).

One can show that if the adversary guesses μ correctly in Game 0 with probability $1/2 + \epsilon$, then we guess the distribution correctly with probability $1/2 + \epsilon/2$.

5. Leveled FHE from LWE

The Gentry-Sahai-Waters (GSW) leveled FHE scheme [20] is currently the conceptually simplest FHE scheme whose security is based on LWE. As a warm-up to build intuition, we first describe how a noise-free (but insecure) version of GSW would work. Then, we introduce noise, describe how to fix the problems it causes, and reduce the security of GSW to the security of Regev's scheme (hence to LWE).

5.1. Thought experiment: Leveled FHE from learning *without* errors. Imagine that Regev's encryption scheme had no error, that an encryption of $\mu \in \{0, 1\}$ is simply a vector $\vec{c} \in (\mathbb{Z}/q\mathbb{Z})^{n+1}$ such that $\langle \vec{c}, \vec{t} \rangle = \mu \in \mathbb{Z}/q\mathbb{Z}$, where $\vec{t}$ is the secret key. How can we add and multiply such ciphertexts so as to add and multiply the plaintexts inside?

Addition is easy. Given two ciphertexts $\vec{c}_1, \vec{c}_2$ that happen to encrypt μ_1, μ_2, we add them to obtain a ciphertext that encrypts the sum: $\langle \vec{c}_1 + \vec{c}_2, \vec{t} \rangle = \mu_1 + \mu_2$.

Multiplication is tricker. We can use tensor products: $\langle \vec{c}_1 \otimes \vec{c}_2, \vec{t} \otimes \vec{t} \rangle = \mu_1 \cdot \mu_2$. However, then each circuit level of multiplications squares the dimension of the ciphertexts, making the scheme non-compact and inefficient.

To get compact multiplication, a better idea is to use matrix multiplication. Specifically, let an encryption of μ be a square matrix C such that $C \cdot \vec{t} = \mu \cdot \vec{t}$. In other words, the secret key is an *eigenvector* of the ciphertext matrix, and the message is the eigenvalue.[3] Addition and multiplication of ciphertexts induces addition and multiplication of plaintexts (eigenvalues). Decryption is a ring homomorphism from the ring of matrices having $\vec{t}$ as an eigenvector to the corresponding eigenvalue.

Unfortunately, this scheme is easy to attack. The encryptions of 0 form a subspace that is easily identified (via linear algebra) once enough encryptions of 0 are collected. More broadly, this eigenvector-based FHE scheme falls within the so-far-unsuccessful *hidden ring homomorphism* approach to FHE. In this approach, the message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ are rings, and decryption $\mathsf{D}_{\mathrm{sk}} : \mathcal{C} \to \mathcal{M}$ is a ring homomorphism that depends on the secret key sk. Addition and multiplication of ciphertexts induce addition and multiplication of plaintexts. Encryptions of 0 form an *ideal* $\mathcal{I}$ in $\mathcal{C}$, while encryptions of 1 are in $1 + \mathcal{I}$. Semantic security relies on the hardness of the *ideal membership problem*: roughly, distinguish whether an element of $\mathcal{C}$ is in $\mathcal{I}$. Another example in this framework is the Polly Cracker scheme proposed by Fellows and Koblitz [13], where the secret key is a secret point in $\vec{s} \in \mathbb{F}_q^n$, and μ is encrypted as a "random" multivariate polynomial that evaluates to μ at $\vec{s}$. Unfortunately, so far, there are no FHE schemes based on hidden ring homomorphisms that are both compact and secure (though the approach has not been ruled out).

5.2. Error-Preserving transformations. As we will see, the GSW scheme uses exactly the above eigenvector approach, but adds noise to it. In GSW, the secret key is a vector $\vec{v}$ with a special form, and an encryption of μ is a matrix C such that $C \cdot \vec{v} = \mu \cdot \vec{v} + \vec{e}$ for small error vector $\vec{e}$ – that is, $\vec{v}$ is an *approximate eigenvector* of the ciphertext, with the message

[3] Note that since we work modulo q, eigenvectors here do not have the usual geometric interpretation.

as the eigenvalue. The noise makes multiplication tricky again, since

$$C_1 \cdot C_2 \cdot \vec{v} = C_1 \cdot (\mu_2 \cdot \vec{v} + \vec{e}_2) = \mu_1 \cdot \mu_2 \cdot \vec{v} + (\mu_2 \cdot \vec{e}_1 + C_1 \cdot \vec{e}_2).$$

The new noise $\mu_2 \cdot \vec{e}_1 + C_1 \cdot \vec{e}_2$ depends not only on the old noises, but also on the second message and the first ciphertext. To ensure that the magnitude of the noise grows at most by a polynomial factor with each circuit level of multiplication, we need to keep the messages small (we do this by restricting messages to $\{0,1\}$ and using NAND gates) and also keep the ciphertexts small.

Here, we describe embarrassingly simple (but very useful) error-preserving transformations that an Evaluator can apply to make the entries of a ciphertext matrix small (in $\{0,1\}$) without knowing or altering what the ciphertext encrypts. The idea is simply to use binary decomposition: we decompose each mod-q coefficient into $\log_2 q$ coefficients in $\{0,1\}$.

Specifically, let $\vec{c}, \vec{t}$ be vectors in $(\mathbb{Z}/q\mathbb{Z})^k$. Let $\ell = \lfloor \log_2 q \rfloor + 1$ and $N = k \cdot \ell$. Let $\mathsf{BitDecomp}(\vec{c}) = (c_{1,0}, \ldots, c_{1,\ell-1}, \ldots, c_{k,0}, \ldots, c_{k,\ell-1})$, a N-dimensional vector where $c_{i,j}$ is the j-th bit in c_i's binary representation, bits ordered least significant to most significant. For $\vec{c}^* = (c_{1,0}, \ldots, c_{1,\ell-1}, \ldots, c_{k,0}, \ldots, c_{k,\ell-1})$, let $\mathsf{BitDecomp}^{-1}(\vec{c}^*) = (\sum 2^j \cdot c_{1,j}, \ldots, \sum 2^j \cdot c_{k,j})$ be the inverse of $\mathsf{BitDecomp}$, but well-defined even when the input is not a 0/1 vector. For N-dimensional $\vec{c}^*$, let $\mathsf{Flatten}(\vec{c}^*) = \mathsf{BitDecomp}(\mathsf{BitDecomp}^{-1}(\vec{c}^*))$, a N-dimensional vector with $0/1$ coefficients. When A is a matrix, let $\mathsf{BitDecomp}(A)$, $\mathsf{BitDecomp}^{-1}(A)$, or $\mathsf{Flatten}(A)$ be the matrix formed by applying the operation to each row of A separately. Finally, let $\mathsf{Powersof2}(\vec{t}) = (t_1, 2t_1, \ldots, 2^{\ell-1}t_1, \ldots, t_k, 2t_k, \ldots, 2^{\ell-1}t_k)$, a N-dimensional vector. Here are some obvious facts:

- $\langle \vec{c}, \vec{t} \rangle = \langle \mathsf{BitDecomp}(\vec{c}), \mathsf{Powersof2}(\vec{t}) \rangle$.
- For any N-dimensional $\vec{c}^*$:
  $\langle \vec{c}^*, \mathsf{Powersof2}(\vec{t}) \rangle = \langle \mathsf{BitDecomp}^{-1}(\vec{c}^*), \vec{t} \rangle = \langle \mathsf{Flatten}(\vec{c}^*), \mathsf{Powersof2}(\vec{t}) \rangle$.

In the GSW scheme, which we finally formally describe in the next subsection, we will use $\vec{v} \leftarrow \mathsf{Powersof2}(\vec{t})$ as the secret key vector, rather than $\vec{t}$. The salient feature of $\mathsf{Flatten}$ is that we can apply it to a matrix C that encrypts a message under $\mathsf{Powersof2}(\vec{t})$ without affecting its product with $\mathsf{Powersof2}(\vec{t})$ and hence what it encrypts, and (importantly) without knowing $\vec{t}$. By $\mathsf{Flatten}$ing ciphertexts after each operation, we ensure that the next operation will increase the magnitude of the error by only a polynomial factor.

5.3. The GSW leveled FHE scheme from LWE. Brakerski and Vaikuntanathan were the first to construct a leveled FHE scheme based on LWE [10]. However, the scheme by Gentry, Sahai and Waters is particularly simple. It uses a "compiler" that transforms any LWE-based public-key encryption scheme $(\mathsf{K}, \mathsf{E}, \mathsf{D})$ that has certain natural properties into a LWE-based leveled FHE scheme $(\mathsf{GSW.K}, \mathsf{GSW.E}, \mathsf{GSW.D}, \mathsf{GSW.NAND})$ capable of Evaluating circuits of NAND gates. Regev's scheme has the needed properties. The properties are:

1. **Property 1 (Vectors and parameters)**: The ciphertext and decryption key are vectors $\vec{c}, \vec{t} \in (\mathbb{Z}/q\mathbb{Z})^{n'}$ for some n'. The first coefficient of $\vec{t}$ is 1 and q is odd.
2. **Property 2 (Small dot product)**: If $\vec{c}$ encrypts 0, then $\langle \vec{c}, \vec{t} \rangle$ is "small".
3. **Property 3 (Security)**: Encryptions of 0 are indistinguishable from uniform vectors over $\mathbb{Z}/q\mathbb{Z}$ (under LWE).

The parameters n, q, χ of the underlying encryption scheme determine the depth L of the circuits that GSW can Evaluate. So, the compiler is not completely black box; K must be tweaked to depend on L. (We will discuss how L affects parameter sizes later.) The GSW scheme works as follows.

- $\mathsf{GSW.K}(1^\lambda, 1^L)$: Compute $\mathsf{K}(1^\lambda, 1^L)$ to obtain parameters $params$, secret vector $\mathrm{sk} = \vec{t} \in (\mathbb{Z}/q\mathbb{Z})^{n'}$ and public key pk. Let $\ell = \lfloor \log q \rfloor + 1$ and $N = n' \cdot \ell$. Set $\vec{v} = \mathsf{Powersof2}(\vec{t})$.
- $\mathsf{GSW.E}(params, \mathrm{pk}, \mu \in \{0,1\})$: Set $\vec{c}_i' \leftarrow \mathsf{E}(params, \mathrm{pk}, 0)$ for i from 1 to N. (*Remark:* These are just N encryptions of 0 under the public key encryption scheme.) Set $C' \in (\mathbb{Z}/q\mathbb{Z})^{N \times n'}$ to be the matrix with rows $\{\vec{c}_i'\}$. Output the ciphertext C given below. (I_N is the N-dimensional identity matrix.)

$$C = \mathsf{Flatten}\big(\mu \cdot I_N + 2 \cdot \mathsf{BitDecomp}(C')\big) \in (\mathbb{Z}/q\mathbb{Z})^{N \times N}.$$

- $\mathsf{GSW.D}(params, \mathrm{sk}, C)$: Let $\vec{c}_1$ be the first row of C. Output $[[\langle \vec{c}_1, \vec{v} \rangle]_q]_2$.
- $\mathsf{NAND}(C_1, C_2)$: To NAND two ciphertexts $C_1, C_2 \in (\mathbb{Z}/q\mathbb{Z})^{N \times N}$, output $\mathsf{Flatten}(I_N - C_1 \cdot C_2)$.

Decryption works, since if C is as above, then

$$\begin{aligned} C \cdot \vec{v} &= (\mu \cdot I_N + 2 \cdot \mathsf{BitDecomp}(C')) \cdot \vec{v} \ \text{[Flatten preserves product with } \vec{v}\text{]} \\ &= \mu \cdot \vec{v} + 2 \cdot C' \cdot \vec{t} \qquad [\mathsf{BitDecomp}(C') \cdot \mathsf{Powersof2}(\vec{t}) = C' \cdot \vec{t}] \\ &= \mu \cdot \vec{v} + 2 \cdot small \qquad \text{[By Property 2 above]}. \end{aligned}$$

Since $v_1 = 1$, the integer $[\langle \vec{c}_1, \vec{v} \rangle]_q = \mu \cdot v_1 + 2 \cdot small$ is small and has the same parity as μ, allowing recovery of $\mu \in \{0,1\}$ when $|small| < q/4 - 1$.

NAND works, since if C_1, C_2 happen to be valid encryptions of $\mu_1, \mu_2 \in \{0,1\}$ with errors $\vec{e}_1, \vec{e}_2$, then:

$$\mathsf{NAND}(C_1, C_2) \cdot \vec{v} = (I_N - C_1 \cdot C_2) \cdot \vec{v} = (1 - \mu_1 \cdot \mu_2) \cdot \vec{v} - \mu_2 \cdot \vec{e}_1 - C_1 \cdot \vec{e}_2$$

Note that NAND maintains the invariant that if the input messages are in $\{0,1\}$, then so is the output message. With this invariant, and using Flatten to ensure that C_1's coefficients are in $\{0,1\}$, the output error is at most $N+1$ times larger than the bigger input error.

Theorem 5.1. GSW *is semantically secure under the LWE assumption.*

Proof. By Property 3, C' is indistinguishable from a uniform matrix under LWE. Thus, since q is odd, $\mathsf{BitDecomp}^{-1}(C) = \mu \cdot \mathsf{BitDecomp}^{-1}(I_N) + 2 \cdot C'$ is indistinguishable from uniform U. But then $C = \mathsf{Flatten}(C)$ is indistinguishable from $\mathsf{BitDecomp}(U)$, where the latter is independent of μ. □

5.4. Parameters and performance. Suppose that we would like to use GSW to evaluate NAND circuits with up to L levels. How should we set the parameters to ensure correctness and security? How much computation does the Evaluate algorithm use per NAND gate?

We have seen that each NAND gate multiplies the magnitude of the error by a factor of at most $N+1$. If β is a bound on the error magnitude of *fresh* ciphertexts, then L levels

of NAND gates amplify the error magnitude to at most $\beta \cdot (N+1)^L$. Decryption works correctly despite such large error, as long as $q/4 - 1 > \beta \cdot (N+1)^L \rightsquigarrow q/\beta > 4N^L$. The ratio q/β must grow exponentially with L to "accommodate" the noise.

Using the rule of thumb that solving GAPSVP_{2^k} in n-dimensional lattices takes time roughly $2^{n/k}$, and acknowledging that a $\text{GAPSVP}_{q/\beta}$ solver would break the scheme, the lattice dimension n (hence N) must increase linearly with $\log(q/\beta)$ to maintain fixed 2^λ security against known attacks. But let us brush this issue under the rug and view n as a fixed parameter. Choosing χ so that β is not too large, and since in practice there is no reason to have $\log q$ grow super-linearly with n, we have $\log q = O(L \log N) = O(L(\log n + \log\log q)) = O(L \log n)$. Given that the NAND procedure is dominated by multiplication of two $N \times N$ matrices for $N = O(n \log q) = \tilde{O}(nL)$, we have the following theorem to characterize the performance of GSW.

Theorem 5.2. *For dimension parameter n and depth parameter L,* GSW *correctly evaluates depth-L circuits of* NAND *gates with $\tilde{O}((nL)^\omega)$ field operations per gate, where $\omega < 2.3727$ is the matrix multiplication exponent.*

Thus, we obtain a leveled FHE scheme with $\text{poly}(\lambda, L)$ computation per NAND gate that achieves 2^λ security against known attacks.

However, even the most theoretical mathematician or computer scientist should be able to see that this scheme will be too slow in practice to Evaluate even moderately complex functions. While LWE-based GSW is far from being the fastest FHE scheme, a big open problem remains: construct a FHE scheme that is truly practical!

As described so far, GSW may leave even a theoretician unsatisfied, as it leaves ample room for qualitative improvement. It begs some questions: Can we make per-gate computation independent of L? Can we Evaluate a priori unbounded depth circuits? Can we actually reduce the noise rather than merely "accommodating" it? For example, can we devise a "refresh" procedure that reduces the noise level of a ciphertext without altering what it encrypts, so that we can Evaluate ad infinitum, refreshing when needed?

The theoretician, at least, may find some solace in the answers we provide in the next section, where we describe precisely such a "refresh" procedure, called *bootstrapping*, that allows Evaluation of unbounded-depth circuits with per-gate computation independent of the depth.

6. Bootstrapping: Homomorphic encryption for unbounded depth circuits

In GSW and all current FHE schemes, ciphertexts are "noisy". Computing over the ciphertexts increases the noise, until eventually the noise becomes bigger than the modulus q, and all hope of reliably decrypting the message correctly is lost. Must we surrender to this life-destroying entropy? Or is there some way to "rejuvenate" an old noisy ciphertext, to create a new ciphertext that encrypts the same value but with much less noise, so that it can safely participate in more computation? Here, we describe a procedure called *bootstrapping* that refreshes ciphertexts, gives them a sort of immortality, so that we can Evaluate unbounded depth circuits with per-gate computation independent of the depth.

6.1. Self-Referentiality in encrypted computation. Can the brain understand itself? Philosophically, it seems appealing to think that, as a brain becomes more complex, so does the

task of understanding it, so that self-understanding remains eternally just out of reach.

Here we consider a somewhat similar question: Can a homomorphic encryption scheme decrypt itself? The decryption function of a homomorphic encryption scheme is, after all, just another function that we can try to plug into the Evaluate algorithm. But does it work? Or, is it the case that, for any L, the decryption function of a leveled FHE scheme capable of Evaluating depth-L circuits has depth greater than L, beyond the Evaluation capacity of scheme?

This is no idle brain-teaser. Actually, among the functions than an FHE scheme can Evaluate, its own decryption function is not only the most interesting, but perhaps also the most useful. Let us consider what we can do with such self-referential encrypted computation. Suppose c encrypts μ under $(\mathrm{pk}, \mathrm{sk})$. Set $\overline{\mathrm{sk}_i} \leftarrow \mathsf{E}(\mathrm{pk}, \mathrm{sk}_i)$ for all of the bits $\{\mathrm{sk}_i\}$ of sk – that is, the ciphertexts $\{\overline{\mathrm{sk}_i}\}$ are an *encryption of the key under itself*. We will publish this encryption of the secret key, so that Evaluators can use it. Set $\overline{c_i} \leftarrow \mathsf{E}(\mathrm{pk}, c_i)$ for all of the bits $\{c_i\}$ of c – that is, these ciphertexts are a *double encryption* of μ. Now, suppose that the leveled FHE scheme can correctly Evaluate L levels, but Evaluating the decryption function D requires only $L-1$ levels. Consider the following ciphertext:

$$c' \leftarrow \mathsf{V}(\mathrm{pk}, \mathsf{D}, (\{\overline{\mathrm{sk}_i}\}, \{\overline{c_i}\})).$$

By the correctness of Evaluate:

$$\mathsf{D}(\mathrm{sk}, c') = \mathsf{D}(\{\mathrm{sk}_i\}, \{c_i\}) = \mu.$$

That is, the new ciphertext c' encrypts the same value as the old ciphertext c. (Interestingly, Evaluating the decryption function on the double encryption $\{\overline{c_i}\}$ removes the *inner* encryption.) Moreover, since c' is the result of Evaluating a circuit of only $L-1 < L$ levels on *fresh* ciphertexts $\{\overline{\mathrm{sk}_i}\}, \{\overline{c_i}\}$, it (possibly unlike c) can be used safely as input to one more NAND gate. Of course, an Evaluator can use this refreshing trick as often as necessary to ensure the noise level of the ciphertexts remains safely bounded. In short, if we have a magical homomorphic encryption scheme capable of Evaluating its own decryption circuit with room to spare, then that homomorphic encryption scheme can be *bootstrapped* into a pure FHE scheme capable of evaluating unbounded depth circuits.[4]

More formally, Gentry [16] defined and proved the following.

Definition 6.1 (Bootstrappable encryption scheme). A homomorphic encryption scheme $\mathcal{E}$ is called *bootstrappable* if $\mathcal{E}$ compactly evaluates all circuits of depth at most $(D+1)$, where D is the depth of $\mathcal{E}$'s decryption circuit, and the computational complexity of $\mathcal{E}$'s algorithms is polynomial in the security parameter and (in the case of the evaluation algorithm) the size of the circuit.

Theorem 6.2 (Bootstrapping Theorem). *For any bootstrappable encryption scheme $\mathcal{E}$, there exists a leveled FHE scheme $\{\mathcal{E}^{(L)}\}$ with related security.*

Letting S be the size of $\mathcal{E}$'s decryption circuit, the per-gate evaluation complexity of the leveled FHE is exactly the complexity of evaluating a $(2S+1)$-gate circuit using the bootstrappable scheme: independent of the depth of the circuit.

Under an assumption of circular security *– that is, an assumption that semantic security is preserved despite publishing an encryption of the secret key under its corresponding public key – one obtains a pure FHE scheme.*

[4] For more intuition, see Gentry's (somewhat dated) 2010 survey [17] on FHE for a full-fledged physical analogy for bootstrapping in terms of gloveboxes inside gloveboxes.

Gentry also provided the first bootstrappable and fully homomorphic encryption schemes based on plausible assumptions.

Circular encryptions sound dangerous, but for most encryption schemes it appears that revealing an encryption of sk under pk does not lead to any attack. On the other hand, it is typically difficult to *prove* that an encryption scheme is circular-secure, hence the need for the additional assumption.

To avoid the circular-security assumption, one can instead provide an *acyclic chain* of encrypted secret keys. One generates a key pair $(\mathrm{pk}_i, \mathrm{sk}_i)$ for each level of the circuit, and provides an encryption of sk_i under pk_{i+1}. In this case, one can prove that the encrypted secret key bits are indistinguishable from encryptions of 0 as long as $\mathcal{E}$ is semantically secure.

6.2. Evaluating the GSW decryption circuit. So, can GSW decrypt itself? It turns out it can, but we need one more trick. The concept of the trick is that, before we bootstrap, we can pre-process the ciphertext into a form that does not permit any more homomorphic operations, but is much less complex to decrypt (and hence to bootstrap).

In more detail, recall that a GSW ciphertext is a matrix, but we use only the first row of the matrix during decryption: $\mu = [[\langle \vec{c}_1, \vec{v} \rangle]_q]_2$. Also, we can use $\vec{t}$ rather than $\vec{v} = \mathsf{Powersof2}(\vec{t})$ as the secret key: $\mu = [[\langle \mathsf{BitDecomp}^{-1}(\vec{c}_1), \vec{t} \rangle]_q]_2$. Now, we only need to decrypt (bootstrap) $\mathsf{BitDecomp}^{-1}(\vec{c}_1)$. However, there is still a problem: the complexity of decrypting it depends on q and hence on L, the number of levels the scheme can Evaluate. Can we remove this dependence, to obtain a ciphertext whose decryption complexity is polynomial in the security parameter λ and completely independent of L? If so, then we are done.

Brakerski and Vaikuntanathan [10] gave a particularly clean way of removing this dependence. They showed that we can apply *modulus reduction* and *dimension reduction* to a Regev-type ciphertext $\vec{c}$ (like our $\mathsf{BitDecomp}^{-1}(\vec{c}_1)$ above), so that the complexity of decrypting the final ciphertext becomes independent of L. Modulus reduction takes an initial ciphertext $\vec{c}$ that encrypts μ modulo q, and outputs a new ciphertext that encrypts μ modulo a smaller modulus p. Dimension reduction reduces the dimension of the ciphertext vector. After applying modulus and dimension reduction, we obtain a ciphertext $\vec{c}^*$ of $\mathrm{poly}(\lambda)$ dimension such that $\mu = [[\langle \vec{c}^*, \vec{t} \rangle]_p]_2$ for small p (e.g., p may even be only polynomial in the security parameter). The size of $\vec{c}^*$ is independent of L.

Let us sketch how modulus reduction works. (We omit a description of dimension reduction.) Recall that Applebaum et al. [2] showed that the hardness of LWE is unaffected when the coefficients of secret key are chosen from the small error distribution χ. When $\vec{t}$ is small and $[\langle \vec{c}_i, \vec{t} \rangle]_q$ is small, then $[\langle \vec{c}_i^*, \vec{t} \rangle]_p$ is also small, where $\vec{c}_i^* = \lfloor (p/q) \cdot \vec{c}_i \rceil$ is simply p/q times $\vec{c}_i$ rounded. The following easy lemma makes this more precise, and also shows that we can preserve other aspects of the noise, such as its parity.

Lemma 6.3. *Let p and q be two odd moduli, and let $\vec{c}$ be an integer vector. Define $\vec{c}^*$ to be the integer vector closest to $(p/q) \cdot \vec{c}$ such that $\vec{c}^* = \vec{c} \bmod 2$. Then, for any $\vec{t}$ with $|[\langle \vec{c}, \vec{t} \rangle]_q| < q/2 - (q/p) \cdot \ell_1(\vec{t})$, we have*

$$\begin{aligned} [\langle \vec{c}^*, \vec{t} \rangle]_p &= [\langle \vec{c}, \vec{t} \rangle]_q \bmod 2 \quad \text{and} \\ |[\langle \vec{c}^*, \vec{t} \rangle]_p| &< (p/q) \cdot |[\langle \vec{c}, \vec{t} \rangle]_q| + \ell_1(\vec{t}) \end{aligned}$$

where $\ell_1(\vec{t}) = \sum |t_i|$ is the ℓ_1-norm of $\vec{t}$.

Proof. For some integer k, we have $[\langle \vec{c}, \vec{t} \rangle]_q = \langle \vec{c}, \vec{t} \rangle - kq$. For the same k, let $e_p = \langle \vec{c}^*, \vec{t} \rangle - kp \in \mathbb{Z}$. Since $\vec{c}^* = \vec{c}$ and $p = q$ modulo 2, we have $e_p = [\langle \vec{c}, \vec{t} \rangle]_q \bmod 2$. To finish the proof, it suffices to prove that $e_p = [\langle \vec{c}^*, \vec{t} \rangle]_p$ and that it has small enough norm. We have $e_p = (p/q)[\langle \vec{c}, \vec{t} \rangle]_q + \langle \vec{c}^* - (p/q)\vec{c}, \vec{t} \rangle$, and therefore $|e_p| \leq (p/q)[\langle \vec{c}, \vec{t} \rangle]_q + \ell_1(\vec{t}) < p/2$. The latter inequality implies $e_p = [\langle \vec{c}^*, \vec{t} \rangle]_p$. □

An alternative view of modulus reduction is that we might as well divide $\vec{c}$ by q and consider its dot product with $\vec{t}$ *modulo 1* – the q merely represents the fact that we represent coefficients of $\vec{c}$ with $\log q$ bits of precision. When we begin to Evaluate a deep circuit, we need lots of precision, since many noise-increasing operations remain. But as we complete the circuit, we can drop precision, allowing the ciphertext to become smaller.

To make a homomorphic encryption scheme bootstrappable, one merely sets the parameters of the scheme so that it is capable of Evaluating the reduced decryption circuit (plus one more NAND gate). The reduced decryption circuit has depth logarithmic in the security parameter. Since each level of NAND gates increases the noise by a polynomial factor, we can bootstrap GSW by setting q to be quasi-polynomial, and (modulo the circular security issue) we can base the security of GSW on LWE for quasi-polynomial factors. Very recently, Brakerski and Vaikuntanathan [11] showed how to go from quasi-polynomial to polynomial by devising a decryption algorithm that, when Evaluated with GSW, increases the noise by only a polynomial factor.

7. Looking beyond bootstrapping

In some sense, the current approach to FHE using noise and bootstrapping has been enormously successful. As we have seen, we can Evaluate arbitrary encrypted functions over encrypted data with overhead only polynomial in the security parameter, independent of the complexity of the function. In fact, we can do even better. We can pack many plaintexts into each ciphertext, and perform batch SIMD (simultaneous instruction multiple data) on encrypted arrays, so as to Evaluate a function many times in parallel without additional computation over many encrypted data-sets [7, 8, 19, 35]. Using a variant of LWE called ring LWE in which the coefficient vectors are over the ring of integers of a cyclotomic number field, we can even move data in encrypted arrays between different array "slots" by using automorphisms of the ring. Using ring LWE with ciphertext-packing and automorphisms, we can get the overhead of FHE down to *polylogarithmic* in the security parameter [19].

Unfortunately, it turns out that polylogarithmic can still be impractically large. The overhead of current FHE schemes is still at least in the high millions for reasonable values of the security parameter. The problem is noise and bootstrapping: Evaluating the decryption circuit after Evaluating each NAND gate in our function seems to inherently require huge overhead, even it is batched to refresh multiple ciphertexts simultaneously. Can we do better? Can we eliminate bootstrapping, or even eliminate noise altogether?

7.1. Can we refresh ciphertexts without bootstrapping? Bootstrapping reduces the noise of a ciphertext by applying Decryption to it inside an Evaluation. But is there a more *direct* way to reduce the noise so as to Evaluate unbounded depth circuits? This is a fascinating open problem.

Quantum error correction (QEC) has a high-level similarity to ciphertext refreshing. To

correct noise in a quantum computation (e.g., phase errors in the qubits), QEC introduces some ancillary bits to the computation, uses them to compute an error correction syndrome over the primary qubits, measures the syndrome, and uses the result to adjust the quantum state of the primary qubits. A peculiarity of QEC is that measurement of the ancillary bits must not reveal anything about the correct values of the primary bits; else, the measurement would collapse the computation. Can we construct an analogous noise reduction technique for FHE, where an Evaluator can compute a syndrome that allows it to reduce ciphertext noise, but still cannot learn what the ciphertext encrypts?

Tao's computational program for Navier-Stokes [36] might be another place to look for new ideas to reduce ciphertext noise. Part of the reason bootstrapping is slow is that it goes "outside of the system": it refreshes a ciphertext not by acting on it directly, but rather by using the ciphertext to construct a function that is Evaluated over fresh encryptions of the secret key bits. If, instead, we could manage ciphertext noise *endogenously* (like Tao's water-based circuits), one could hope that eliminating the layer of indirection would also reduce computational complexity.

7.2. Can we eliminate noise altogether? The noisiness of LWE-based ciphertexts is the basis of their security, but also an obstacle to making FHE practical. Can we construct an FHE scheme without noise?

Without noise, decryption in GSW is a purely linear function, and the system can be broken easily using linear algebra. More generally, for any encryption scheme in which $\mathsf{D}(\mathrm{sk}, c)$ is a degree k polynomial, we can view $\mathsf{D}(\mathrm{sk}, c)$ as a dot product $\langle M(\mathrm{sk}), M(c)\rangle$ of the vectors of monomials of degree at most k associated to sk and c, and an attacker can use linear algebra to break semantic security in time $\lambda^{O(k)}$. So, to get 2^λ security, the degree of $\mathsf{D}(\mathrm{sk}, c)$ must be essentially linear in λ, a "complex" function. And yet, for an FHE scheme, $\mathsf{D}(\mathrm{sk}, c)$ must also be robust and flexible enough to allow computation.

Interestingly, the noise in LWE-based schemes boosts the degree of the decryption function. Although $[[\vec{c}, \vec{t}]_q]_2$ looks "almost linear", the rounding makes it high degree both modulo q and modulo 2. On the other hand, the "almost linearity" of decryption allows computation.

As some some final food for thought, we sketch an interesting but so-far-unrealized framework due to Nuida [28] for constructing pure noise-free FHE using non-abelian groups. Unfortunately, the framework also illustrates the difficulty of avoiding linear algebra attacks, even in contexts (using groups rather than rings) where one might hope they are inapplicable. First, a couple of definitions:

Definition 7.1 (Perfect Group Pairs). We call (G, H) a *perfect group pair* if G and H are both finite perfect groups (equal to their commutator subgroups) and H is a normal subgroup of G. We also require that G and H have efficient ($\mathrm{polylog}(|G|)$) operations – in particular, given a set of group generators of G or H, one can re-randomize them to obtain a random set of B group elements (for some polynomial bound B) that generate the same group.

Definition 7.2 (Perfect Group Pair Decision (PGP) Problem). Given (generators for) a perfect group pair G and H, and a third set of generators that generates G or H, distinguish which.

The form of the ciphertexts is simple: an encryption of 1 is a set of generators of G, while an encryption of 0 is a set of generators of H. The public key contains encryptions of '1' and '0' that the encrypter can randomize to generate its ciphertext. Decryption will

use some (unspecified) secret key $\tau_{G,H}$ that allows the keyholder to distinguish between generators for G and H. Semantic security follows directly from the PGP assumption and the re-randomizability of the group generators.

We describe homomorphic operations only for AND and OR gates (monotone circuits). Suppose the inputs to the gate are generators of (unknown) groups K_1, K_2. To Evaluate an OR gate, output (randomized) generators for the join of K_1 and K_2. (The output group is G iff an input group is G and H otherwise, and thus computes OR correctly.) To Evaluate an AND gate, output (randomized) generators for the commutator $[K_1, K_2]$. (Since H is normal in G, the output group is H iff an input group is H and G otherwise, and thus computes AND correctly.)

The main open problem for this framework is to find suitable perfect group pairs. It is easy to find perfect group pairs for which the PGP problem is easy: for example, take $G = H \times K$ for perfect groups H and K, where the extra coordinate makes elements of G easy to identify. Also, there are various perfect matrix group pairs (G, H) where the PGP problem is less trivial, but still ultimately solvable via linear algebra. Even if the groups are not initially presented as matrices, one must avoid groups with efficiently computable representations that enable linear algebra attacks. Still, this framework, though unrealized, serves as a useful counterpoint to the notion that noise and bootstrapping may be necessary to obtain FHE.

References

[1] Daniel Apon, Xiong Fan, Jonathan Katz, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou, *Non-interactive cryptography in the ram model of computation*, IACR Cryptology ePrint Archive, 2014:154, 2014.

[2] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai, *Fast cryptographic primitives and circular-secure encryption based on hard learning problems*, In CRYPTO, Springer, 2009, pp. 595–618.

[3] Sanjeev Arora and Rong Ge, *New algorithms for learning in presence of errors*, In ICALP, Springer, 2011, pp. 403–415.

[4] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang, *On the (im)possibility of obfuscating programs*, In CRYPTO, Springer, 2001, pp. 1–18.

[5] Dan Boneh, Amit Sahai, and Brent Waters, *Functional encryption: Definitions and challenges* In TCC, Springer, 2011, pp. 253–273.

[6] Zvika Brakerski, *Fully homomorphic encryption without modulus switching from classical gapsvp*, In CRYPTO, Springer, 2012, pp. 868–886.

[7] Zvika Brakerski, Craig Gentry, and Shai Halevi, *Packed ciphertexts in lwe-based homomorphic encryption* In PKC, Springer, 2013, pp. 1–13.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, *(leveled) fully homomorphic encryption without bootstrapping*, In ITCS, ACM, 2012, pp, 309–325.

[9] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé, *Classical hardness of learning with errors*, In STOC, ACM, 2013, pp. 575–584.

[10] Zvika Brakerski and Vinod Vaikuntanathan, *Efficient fully homomorphic encryption from (standard) LWE*, In FOCS, IEEE, 2011, pp. 97–106.

[11] Zvika Brakerski and Vinod Vaikuntanathan, *Lattice-based fhe as secure as pke*, In ITCS, ACM, 2014, pp. 1–12.

[12] Whitfield Diffie and Martin E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22**(6) (1976), 644–654.

[13] Michael Fellows and Neal Koblitz, *Combinatorial cryptosystems galore!*, Contemporary Mathematics **168** (1994), 51–51.

[14] Sanjam Garg, Craig Gentry, and Shai Halevi, *Candidate multilinear maps from ideal lattices*, In EUROCRYPT, Springer, 2013, pp. 1–17.

[15] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters, *Candidate indistinguishability obfuscation and functional encryption for all circuits*, In FOCS, IEEE, 2013, pp. 40–49.

[16] Craig Gentry, *Fully homomorphic encryption using ideal lattices* In STOC, ACM, 2009, pp. 169–178.

[17] ———, *Computing arbitrary functions of encrypted data*, Commun. ACM, **53**(3) (2010), 97–105.

[18] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs, *Outsourcing private ram computation*, IACR Crypt. ePrint Arch., 2014:148, 2014.

[19] Craig Gentry, Shai Halevi, and Nigel P. Smart, *Fully homomorphic encryption with polylog overhead*, In EUROCRYPT, Springer, 2012, pp. 465–482.

[20] Craig Gentry, Amit Sahai, and Brent Waters, *Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based*, In CRYPTO, Springer, 2013, pp. 75–92.

[21] Shafi Goldwasser and Silvio Micali, *Probabilistic encryption and how to play mental poker keeping secret all partial information*, In STOC, ACM, 1982, pp. 365–377.

[22] Shafi Goldwasser and Guy N. Rothblum, *On best-possible obfuscation*, In TCC, Springer, 2007, pp. 194–213.

[23] Russell Impagliazzo, *A personal view of average-case complexity*, In Structure in Complexity Theory Conference, IEEE, 1995, pp. 134–147.

[24] Joe Kilian, *Founding cryptography on oblivious transfer*, In STOC, ACM, 1988, pp. 20–31.

[25] Silvio Micali, 1988, Personal communication to Joe Kilian in [24].

[26] Daniele Micciancio and Petros Mol, *Pseudorandom knapsacks and the sample complexity of lwe search-to-decision reductions*, In CRYPTO, Springer, 2011, pp. 465–484.

[27] Daniele Micciancio and Chris Peikert, *Trapdoors for lattices: Simpler, tighter, faster, smaller* In EUROCRYPT, Springer, 2012, pp. 700–718.

[28] Koji Nuida, *A simple framework for noise-free construction of fully homomorphic encryption from a special class of non-commutative groups*, IACR Cryptology ePrint Archive, 2014:097, 2014.

[29] Chris Peikert, *Public-key cryptosystems from the worst-case shortest vector problem: extended abstract*, In STOC, ACM, 2009, pp. 333–342.

[30] Nicholas Pippenger and Michael J Fischer, *Relations among complexity measures*, Journal of the ACM (JACM) **26**(2) (1979), 361–381.

[31] Oded Regev, *On lattices, learning with errors, random linear codes, and cryptography*, In STOC, ACM, 2005, pp. 84–93.

[32] Ron Rivest, Leonard Adleman, and Michael Dertouzos, *On data banks and privacy homomorphisms*, In Found. of Sec. Comp., 1978, pp. 169–180.

[33] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Commun. ACM **21**(2) (1978), 120–126.

[34] Claus-Peter Schnorr, *A hierarchy of polynomial time lattice basis reduction algorithms* Theor. comp. sci. **53**(2) (1987), 201–224.

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

[36] Terence Tao, *Finite time blowup for an averaged three-dimensional navier-stokes equation*, arXiv:1402.0290, 2014.

IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
E-mail: cbgentry@us.ibm.com