Chapter 5
Homomorphic Encryption

Shai Halevi

Abstract Fully homomorphic encryption (FHE) has been called the "Swiss Army knife of cryptography", since it provides a single tool that can be uniformly applied to many cryptographic applications. In this tutorial we study FHE and describe its different properties, relations with other concepts in cryptography, and constructions. We briefly discuss the three generations of FHE constructions since Gentry's breakthrough result in 2009, and cover in detail the third-generation scheme of Gentry, Sahai, and Waters (GSW).

5.1 Computing on Encrypted Data

Secure multiparty computation epitomizes the promise of cryptography, performing the seemingly impossible magic trick of processing data without having access to it. One simple example features a client holding an input x and a server holding a function f, the client wishing to learn $f(x)$ without giving away information about its input. Similarly, the server may want to hide information about the function f from the client (except, of course, the value $f(x)$). This situation arises in many practical scenarios, most notably in the context of secure cloud computing; For example, the client may want to get driving directions without revealing their location to the server.

Cryptographers have devised multiple solutions to this problem over the last 40 years, but none simpler (conceptually) than the paradigm of *computing on encrypted data*. This paradigm was suggested by Rivest et al. [81] in the very early days of public-key cryptography, under the name "privacy homomorphisms": The client simply encrypts its input x and sends the ciphertext to the server, who can "evaluate the function f on the encrypted input". The server returns the evaluated ciphertext to the client, who decrypts it and recovers the result. Of course, it takes a special encryption method to allow such processing of encrypted data; for example, Rivest

Shai Halevi
IBM T.J. Watson Research Center, Yorktown Heights, USA, e-mail: shaih@alum.mit.edu

Y. Lindell (ed.), *Tutorials on the Foundations of Cryptography*,
Information Security and Cryptography, DOI 10.1007/978-3-319-57048-8_5

et al. observed in [81] that “raw RSA” (where x is encrypted as $x^e \bmod N$) enables multiplication of encrypted values. They asked whether it was possible to compute more general functions on encrypted data, and what one can do with an encryption scheme that enables such computations.

Following [81], encryption schemes that support computation on encrypted data came to be known as *homomorphic encryption* (HE). In addition to the usual encryption and decryption procedures, these schemes have an *evaluation procedure* that takes ciphertexts encrypting x and a description of a function f, and returns an “evaluated ciphertext” that can be decrypted to obtain the value $f(x)$. A salient non-triviality property of such scheme is *compactness*, requiring that the complexity of decrypting an evaluated ciphertext does not depend on the function f that was used in the evaluation. Another desirable security property is *function-privacy*, requiring that the evaluated ciphertext does not reveal the function f, even to the owner of the secret key.

Many cryptosystems that support computation of *some* functions on encrypted data have been proposed over the years, but it seemed much harder to construct a compact *fully homomorphic encryption* (FHE), namely a compact scheme that can evaluate *all* (efficient) functions. It was not until 2009 that the watershed work of Gentry [47] established for the first time a blueprint for constructing such schemes and described a viable candidate. That work was followed by a sequence of rapid advancements, resulting in much more efficient FHE schemes under well established hardness assumptions, and better understanding of the relations between FHE and other branches of secure computation. The goal of this tutorial is to present an overview of that line of work.

A “paradox” and its resolution. The ability to compute on encrypted data may seem paradoxical at first glance. Consider a simple example of outsourced storage: The client stores multiple encrypted files at the server, and later wants to retrieve one of them without the server learning which one was retrieved. The client can send an encrypted index to the server, and the server will perform the computation on the encrypted index and files, resulting in an encryption of only the file that needs to be retrieved.

But the server can still see the encrypted versions of all the stored files, and now it has the encrypted version of the file that the client wants to retrieve. Can’t the server just “see” which of the stored encrypted files is the one being retrieved? The resolution, of course, is that we require the encryption to be *semantically secure* [52], and in particular it must be randomized. Hence there are many different encrypted versions of each file, and the encrypted file that the server returns is different from and “seemingly unrelated” to the stored encrypted files.

5.1.1 Applications of Homomorphic Encryption

Fully homomorphic encryption has been called the “Swiss Army knife of cryptography” [6], since it provides a single tool that can be uniformly applied in a wide host of applications. Even when we have other solutions to a cryptographic problem, the

FHE-based ones are often conceptually simpler and easier to explain. Below we list a few example applications that demonstrate the power of FHE.

Outsourcing storage and computations. Perhaps the most direct application of homomorphic encryption is for outsourcing storage and computation without revealing sensitive information. Consider a small company trying to move its computing facilities to the cloud, but that is wary of the cloud provider having access to the company's confidential information. An easy solution is to encrypt the confidential information before storing it in the cloud, but how can the company use that information without shipping it back on-premises for every operation (which would defeat the purpose of outsourcing it)? Homomorphic encryption provides an elegant solution to this conundrum. The company can keep the information in the cloud in encrypted form, and the cloud provider can process the information in this form and send only the processed result back to the company to be decrypted.

PIR and other private queries. Another direct application of homomorphic encryption is to enable private queries to a database or a search engine. The simplest such example is *private information retrieval* [24], where a server is holding a large database (e.g., the US patent database), and a client wants to retrieve one record of this database without the server learning which record was retrieved. Homomorphic encryption lets the user encrypt the index of the record that it wants to retrieve. The server can evaluate the function $f_{db}(i) = db[i]$ on the encrypted index,[1] returning the encrypted result to the client, who can decrypt it and obtain the plaintext record.

The same solution applies also to the more complex settings of an SQL query to a database or a free-form query to a search engine. In both cases the server has some procedure for handling queries, which can be formalized as a function $g_{db}(\mathsf{query}) = \mathsf{answer}$. The client can therefore encrypt its query before sending it to the server, and the server can evaluate g_{db} on the encrypted query and return the encrypted answer.

General two-party computations. The examples above are special cases of (two-party) secure computation, where two mutually suspicious parties want to compute a common function on their joint input. Specifically we have Alice with input x and Bob with input y, and we want Alice to learn $F(x, y)$ (and nothing more), for some agreed-upon function F, and Bob should learn nothing. In the semi-honest adversarial model (where both parties are assumed to follow the prescribed protocol), this can be achieved by Alice encrypting her input x under her own key, and Bob evaluating the function $F_y(x) = F(x, y)$ on that encrypted input. By the semantic security of the encryption scheme, we know that Bob does not learn anything about Alice's input from the encryption that he sees. Also, if we use a homomorphic scheme that hides the evaluated function, then Alice does not learn anything other than the value of $F(x, y)$. (Converting this protocol to the more general malicious-adversary model can be done using standard techniques [50].)

[1] The function f_{db} has the database db hard wired in its description, and on input i it outputs the i-th record.

We note that all the examples so far can use *secret-key* homomorphic encryption. However, as we will see later in Section 5.2.2.6, for homomorphic encryption the distinction between public-key and secret-key encryption is immaterial.

Zero-knowledge. Homomorphic encryption can also be used in a very simple zero-knowledge proof protocol for every language L in NP. Let $R_L(x, w)$ be an NP relation defining the language $L = \{x : \exists w \text{ s.t. } R_L(x, w) = 1\}$, and we sketch a protocol (taken from [6]) by which Bob who knows w can prove to Alice that $x \in L$.

Toward a protocol, let Bob encrypt its witness w and send it to Alice, who can evaluate on the encrypted witness the function $r_x(w) = R_L(x, w)$. But Alice only has the encrypted result; she still needs Bob's help in determining the bit which is encrypted there. Of course Alice cannot just send that encryption to Bob to be decrypted, since Bob cannot be trusted to return the right answer. Instead, she chooses a random bit b, and sends the evaluated ciphertext to Bob only when $b = 1$, otherwise sending him a fresh encryption of zero. Alice finally accepts if Bob replies with the bit b; otherwise she rejects.

The soundness of this protocol follows from the fact that, for $x \notin L$, both ciphertexts will be encryptions of zero. As long as Bob cannot distinguish fresh from evaluated encryption of zero, he cannot convince Alice with probability better than 1/2. Of course we should consider what happens when the initial ciphertext sent by Bob is not a valid encryption at all, but this can be handled by simple cut-and-choose: First, Bob generates two public keys and Alice asks him to open one of them to show that it was generated correctly. Then, using the other unopened key, Bob encrypts two random strings whose XOR is the witness w, and Alice asks that he opens one set and prove that it was encrypted correctly. It is not hard to show that the resulting protocol is sound, in that for $x \notin L$ no cheating strategy can convince Alice with probability much better than 7/8.

This protocol as described is only *honest-verifier* zero knowledge, since a cheating Alice can send to Bob (say) the first bit of the encrypted witness, rather than the result of evaluating $r_x(\cdot)$. This can be fixed by using a standard commitment technique, where Bob sends to Alice not the decrypted bit itself but rather a commitment to that bit. Then Alice reveals her randomness, demonstrating to Bob that she ran the computation as needed, and only then does Bob open its commitment.

5.1.2 Beyond Homomorphic Encryption

Versatile as it is, homomorphic encryption of course does not solve every problem in cryptography. Some limitations of homomorphic encryption are listed below, and are discussed in more detail in Section 5.5.4.

- The output is encrypted. Although we can evaluate arbitrary functions on encrypted data, the outcome of such computation is itself a ciphertext, and no one can make sense of it without the secret key. In contrast, *obfuscation* and *functional encryption* allow some forms of encrypted computation in which the output is obtained in the clear; see Section 5.5.4.3.

- All inputs must be encrypted under the same key. To be able to compute on encrypted data, all of that data must be encrypted under the same key. Extensions of homomorphic encryption that can process together encrypted data under multiple keys are discussed in Section 5.5.4.1.
- No integrity guarantees. While homomorphic encryption enables computing on encrypted data, it does not offer any way of checking that the computation was indeed carried out as expected. Typically there is no way to tell if a given ciphertext is indeed the result of running some computation or just a fresh encryption of the same value. See Section 5.5.4.2 for more discussion.

5.1.3 Abridged History

Techniques that enable *non-compact* fully homomorphic encryption (where the size of the ciphertext grows with the complexity of the evaluated function) go back to the early 1980s. In particular, as we explain in Section 5.2.2.4, such schemes can be constructed using secure computation techniques such as Yao's garbled circuits [90]. Also known since the 1980s are compact additively homomorphic or multiplicatively homomorphic schemes, i.e., compact schemes that support only addition or only multiplication on encrypted data. Examples of such schemes include Goldwasser–Micali [52] and ElGamal [33].

Going beyond one-operation homomorphism took longer. Boneh, Goh, and Nissim described in 2005 a cryptosystem that permitted an arbitrary number of additions and one multiplication, without growing the ciphertext size [10]. The security of that scheme was based on the hardness of the subgroup-membership problem in composite-order groups that admit bilinear maps. A scheme with similar characteristics was described by Gentry et al. under the learning-with-errors assumption [47]. In 2007, Ishai and Paskin described a compact scheme that can evaluate branching programs [62], with security under the N-th residuosity assumption [77]. Also Melchor et al. [69] described a template for constructing encryption schemes that can evaluate shallow circuits, where the ciphertext size grows exponentially with the multiplication depth but additions are supported without increasing the size, and realizations of that template are known from various lattice hardness assumptions [47, 69].

5.1.3.1 Three Generations of FHE

The first plausible construction of FHE was given by Gentry in 2009 [47]. The development of FHE since that result can be roughly partitioned into three "generations". The first generation includes Gentry's original scheme using ideal lattices [47], and the somewhat simpler scheme of van Dijk et al. that uses only integer arithmetic [88]. Both these schemes suffered from a problem of rapidly growing noise, which affected both efficiency and security (see below). The second generation began in 2011 with the works of Brakerski–Vaikuntanathan [19] and Brakerski et al. [16], and was characterized by much better techniques for controlling the noise, resulting in improved efficiency while at the same time basing security on well-established hardness assumptions. These techniques were accompanied by methods for improving the plaintext-to-ciphertext expansion ratio, further improving efficiency. The third generation began with the scheme of Gentry et al. [48], exhibiting a somewhat different noise development pattern. The third-generation schemes are in general somewhat less efficient than second-generation ones, but they can be based on somewhat weaker hardness assumptions. Below we briefly sketch these three generations, and in Sections 5.3 and 5.4 we describe in detail the third-generation GSW scheme from [48].

Gentry's blueprint. In his celebrated work from 2009 [47], Gentry gave a blueprint for realizing fully homomorphic encryption and described a concrete realization of this blueprint. Gentry's realization employed ideal lattices and assumed the hardness of the "ideal coset problem" in such lattices. In his PhD thesis [37], Gentry also sketched a simpler construction (due to van Dijk) using integer arithmetic; that construction was later completed and analyzed by van Dijk, Gentry, Halevi and Vaikuntanathan in [88]. Here we use the van Dijk et al. construction (viewed as a secret-key encryption scheme) to illustrate the ideas in Gentry's blueprint.

The secret key of the integer-based scheme is a secret large odd integer p, and ciphertexts are integers that are *close to a multiple of p*. An encryption of a bit $b \in \{0, 1\}$ is an integer whose residue modulo p has the same parity as the plaintext bit b. Namely, $c = pq + 2r + b$, where the integers q, r are random and $|r| \ll p$. To decrypt such a ciphertext, first reduce it modulo p into the symmetric interval $[-p/2, p/2)$ and then output the parity of the result, $b = (c \bmod p) \bmod 2$.

Consider what happens when you add or multiply two such ciphertexts. Let $c_i = q_i p + 2r_i + b_i$ for $i = 1, 2$, and denote $c^+ = c_1 + c_2$ and $c^\times = c_1 \cdot c_2$. Then both c^+ and $c^\times$ have similar structure to the original one, namely

$$\begin{aligned} c^+ &= (q_1 + q_2)p + 2(r_1 + r_2) + (b_1 + b_2) \;=\; q'p + 2r' + (b_1 \oplus b_2) \\ \text{and } c^\times &= (q_1 c_2 + c_1 q_2)p + 2(b_1 r_2 + r_1 b_2 + 2r_1 r_2) + b_1 b_2 \;=\; q''p + 2r'' + b_1 b_2 \end{aligned}$$

for some q', q'', r', r'' (where $r' \approx r_1 + r_2$ and $r'' \approx 2r_1 r_2$). If the initial noise quantities r_1, r_2 were small enough relative to p then the new c^+ and $c^\times$ are still decryptable to the right values. More generally it is possible to compute low-degree polynomials on ciphertexts, which will be decrypted to the evaluation of the same polynomials (over $\mathbb{Z}_2$) applied to the plaintext bits. It was proved in [88] that this scheme is secure if the approximate-GCD problem is hard (which is essentially the problem of finding p), and we believe that approximate-GCD is indeed hard for appropriate parameters.

As described, the scheme above is not compact since the bit size of the ciphertext grows with the degree of the evaluated polynomial, but van Dijk et al. described in [88] a way of fixing it by publishing many near multiples of p of various sizes and using modular reduction. A more severe problem, however, is that the *homomorphic capacity* of the scheme is limited to low-degree polynomials (i.e., these are the only functions that it can evaluate). The reason is that the noise magnitude grows rapidly with the degree, until it becomes larger than $p/2$ and then decryption fails. The same noise development problem occurs also in Gentry's construction from [47], and indeed in every FHE construction since.

Bootstrapping. Gentry solved the problem of limited homomorphic capacity by introducing *bootstrapping*. He observed that any homomorphic scheme which is capable of evaluating its own decryption circuit (plus a single NAND gate) can be turned into a fully homomorphic scheme. Specifically, for every two ciphertexts c_1, c_2, consider the function

$$D^*_{c_1,c_2}(\mathsf{sk}) \overset{def}{=} NAND(\mathsf{Decrypt}(\mathsf{sk}, c_1), \mathsf{Decrypt}(\mathsf{sk}, c_2)).$$

Namely, this function takes as input an alleged secret key, uses it to decrypt the two fixed ciphertexts c_1, c_2, and outputs the NAND of the two resulting bits. If the homomorphic capacity of the scheme suffices to evaluate the functions $D^*_{c_1,c_2}(\mathsf{sk})$ for every two ciphertexts c_1, c_2, then the scheme is called *bootstrappable*, and it can be transformed into an FHE scheme, as follows: Publishing an encryption of the secret key sk under the public key, we can homomorphically compute the NAND of any two ciphertexts by evaluating the function $D^*_{c_1,c_2}$ on the publicly available encryption of sk.

Importantly, the homomorphic computation is only applied to the fresh encryption of sk; the ciphertexts c_1, c_2 are only used to *define the function* $D^*_{c_1,c_2}$ that we need to evaluate. We have the guarantees that, as long as c_1, c_2 are decryptable then so is the result of the homomorphic NAND, and the process can be repeated as many times as needed. Thus, we obtain a compact fully homomorphic encryption.

The first generation of FHE schemes. The first generation of FHE constructions included the schemes from [47, 88] and some variations [85, 41, 27, 40]. The main problem with those schemes was the very rapid growth of noise, which severely limited their homomorphic capacity. Specifically, starting from fresh encryptions with noise magnitude ρ, evaluating a degree-d polynomial resulted in noise magnitude of roughly ρ^d. Although these schemes feature a shallow $NC1$ decryption, still the rapid noise growth prevents them from evaluating their own decryption procedure, and additional complex transformations (involving ad hoc hardness assumptions) are needed to enable bootstrapping.

Second-generation FHE. In 2011, a sequence of works by Brakerski et al. [19, 16, 14] developed new techniques for better noise control. These techniques (some of which we describe in Section 5.3.1.4) slowed the growth of noise dramatically, roughly from linear to logarithmic in the degree of the evaluated function. This resulted in "leveled" schemes that can evaluate circuits of any *fixed* polynomial depth, as well as in bootstrappable schemes that can be made fully homomorphic. Moreover, the security of these new schemes could be based on more standard hardness assumptions such as learning with errors (which we describe in Section 5.3.1.1). Some other second-generation schemes (based on the NTRU hardness assumption) were described in [67, 13].

Other techniques were developed for improving the efficiency of homomorphic evaluation, such as "packing" many plaintext bits in a single ciphertext [86, 17, 44] and various bootstrapping optimizations [43, 2, 57]. These optimizations resulted in schemes whose asymptotic overhead (versus computing in the clear) is only polylogarithmic in the security parameter, and with vastly improved practical performance [45, 56]. Second-generation schemes and their optimizations are not covered in detail in this tutorial, but we briefly touch on the optimizations in Section 5.5.1.

The GSW scheme and third-generation FHE. In 2013, Gentry et al. described in [48] yet another homomorphic encryption scheme, which has somewhat different flavor than the second-generation schemes above (and is arguably conceptually simpler). In particular, that scheme had *asymmetric multiplication*, in the sense that the homomorphic multiplication $c_1 \otimes c_2$ results in a different ciphertext than $c_2 \otimes c_1$ (both of which encrypt the same product $b_1 \cdot b_2$). More importantly, the noise growth is also asymmetric: the noise in the left multiplicand has greater influence on the result than the noise in the right multiplicand.

Brakerski and Vaikuntanathan observed in [20] that this asymmetry can be used to obtain even slower rates of noise growth, by designing circuits in which the left multiplicand always has small noise. This observation enabled further reduction of parameters and quantitative relaxation of the underlying hardness assumption. However, this technique is not compatible with some of the performance optimizations in second-generation schemes, and third-generation schemes have higher overhead than their second-generation counterparts.

In this tutorial we cover the GSW scheme in detail, starting in Section 5.3 from the basic leveled scheme, then showing in Section 5.4 how to use bootstrapping to turn it into a fully homomorphic scheme, and how to use the asymmetric noise development to improve parameters and relax the hardness assumption.

5.1.4 Organization of This Tutorial

In Section 5.2, we define homomorphic encryption and its properties and discuss the relations between these properties and connections with secure computation protocols. In Sections 5.3 and 5.4, we describe the GSW construction in detail, and in Section 5.5, we briefly go over many related topics, and in Section 5.6 we suggest further reading.

5.2 Defining Homomorphic Encryption

5.2.1 Notations and Basic Definitions

For an integer $q \in \mathbb{N}$, we identify the quotient group $\mathbb{Z}_q$ with its representatives in the symmetric interval $[-\frac{q}{2}, \frac{q}{2})$ (except $\mathbb{Z}_2$ which is identified with $\{0, 1\}$). For a real number $x \in \mathbb{R}$, we denote by $[x]_q$ the reduction of x modulo q into the same symmetric interval. $\lceil x \rfloor$ is the rounding of x to the nearest integer, and $\lfloor x \rfloor, \lceil x \rceil$ are the floor and ceiling functions. All these notations extend naturally to vectors and matrices element-wise. The inner product of two vectors $\mathbf{u}, \mathbf{v}$ is denoted $\langle \mathbf{u}, \mathbf{v} \rangle$.

For a random process such as running a probabilistic Turing machine M on input x, we write $M(x)$ or $y \leftarrow M(X)$ to describe a random variable which is drawn from the output space of that process. We sometimes write $y := M(x; r)$ to separate the randomness used and the deterministic processing of that randomness. We often use the same notation for a distribution and its support set, so $y \in M(x)$ means that y is a string which is output by $M(x)$ with non-zero probability. Conversely, if S is a set, then $x \leftarrow S$ is a random variable uniformly distributed in S. The ℓ_1 statistical distance between two distributions D_1, D_2 is denoted $SD(D_1, D_2)$.

We denote the output distribution of a multistep process, or the probability of an event over that distribution, using the syntax

$$\text{Distribution} = \{\text{output} : \text{Process}\}, \text{ or } \Pr[\text{event} : \text{Process}].$$

For example, the distribution of encryptions of zero in some cryptosystem can be written as $\{c : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0)\}$, and the probability of an adversary outputting 1 on a ciphertext from this distribution is denoted $\Pr[A(c) = 1 : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0)]$.

5.2.1.1 Homomorphic Encryption

To define *computation* on encrypted data, we must fix some model of computation. In this tutorial we always use binary circuits with fan-in-2 gates to describe the functions that we want to compute on encrypted data.[2] For a binary circuit Π, we denote its size (i.e., number of gates) by $\mathsf{size}(\Pi)$, its depth by $\mathsf{depth}(\Pi)$ (i.e., longest input-to-output path), and the number of inputs by $\mathsf{inpLen}(\Pi)$. We briefly discuss homomorphic encryption relative to other models of computation in Section 5.5.3. On the other hand, we use somewhat informal terms such as algorithm/procedure (or adversary) when referring to computations that manipulate ciphertexts and keys. Unless stated otherwise, all the algorithms/procedures/adversaries below are (uniform) probabilistic polynomial time (PPT).

A homomorphic public-key encryption scheme (for plaintext space $\mathcal{M} = \{0, 1\}$) has four PPT procedures: the usual KeyGen, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$, and also an $\mathsf{Evaluate}$ procedure for computing on encrypted data. We slightly extend the standard syntax of key generation, allowing it to depend not only on the security parameter λ but also on a second "functionality parameter" τ. Throughout this tutorial, some schemes will make use of that parameter while others will not.

Definition 5.2.1 (Syntax). *A homomorphic encryption scheme consists of four procedures, $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$:*

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau)$. *Takes the security parameter λ and another parameter τ and outputs a secret/public key-pair.*[3]

[2] Fixing circuits with binary input also means that we consider bit-encryption schemes, where the plaintext space is $\mathcal{M} = \{0, 1\}$.

[3] We assume that the size of the public and secret keys is set deterministically by λ and τ; i.e., it does not vary with the randomness of the key generation procedure.

- $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, b)$. *Given the public key and a plaintext bit, outputs a ciphertext.*
- $b \leftarrow \mathsf{Decrypt}(\mathsf{sk}, c)$. *Given the secret key and a ciphertext, outputs a plaintext bit.*
- $\mathbf{c}' \leftarrow \mathsf{Evaluate}(\mathsf{pk}, \Pi, \mathbf{c})$. *Takes a public key* pk*, a circuit Π, a vector of ciphertexts $\mathbf{c} = \langle c_1, \ldots, c_t \rangle$, one for every input bit of Π, and outputs another vector of ciphertexts $\mathbf{c}'$, one for every output bit of Π.*

The syntax from Definition 5.2.1 is naturally extended to vectors of plaintexts and ciphertexts. Namely, we write $\mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b})$, where $\mathbf{b} = (b_1, \ldots, b_t)$ and $\mathbf{c} = (c_1, \ldots, c_t)$, to denote $c_i \leftarrow \mathsf{Encrypt}(\mathsf{pk}, b_i)$ for all i. Similarly, we write $\mathbf{b} \leftarrow \mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$ to denote $b_i \leftarrow \mathsf{Decrypt}(\mathsf{sk}, c_i)$ for all i.

We sometimes refer to ciphertexts output by Encrypt as "fresh ciphertexts", and those output by Evaluate are "evaluated ciphertexts". The correctness condition below refers to both.

Definition 5.2.2 (Correctness). *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ be a homomorphic encryption scheme and $\mathcal{C} = \{\mathcal{C}_\tau\}_{\tau \in \mathbb{N}}$ be some circuit family. $\mathcal{E}$ is* (perfectly) correct *for $\mathcal{C}$ if it correctly decrypts both fresh and evaluated ciphertexts. Namely, for all $\lambda, \tau \in \mathbb{N}$, the following two conditions hold:*

- *For any $b \in \{0, 1\}$,*

$$\Pr\Big[\mathsf{Decrypt}(\mathsf{sk}, c) = b : (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, b)\Big] = 1;$$

- *For every $\Pi \in \mathcal{C}_\tau$ and plaintext bits $\mathbf{b} = (b_1, \ldots, b_t) \in \{0, 1\}^t$, one for every input bit of Π,*

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c}') = \Pi(\mathbf{b}) : \begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}),\ \mathbf{c}' \leftarrow \mathsf{Evaluate}(\mathsf{pk}, \Pi, \mathbf{c}) \end{array}\right] = 1.$$

We sometimes refer informally to the largest family $\mathcal{C}$ for which $\mathcal{E}$ is correct as the *homomorphic capacity* of $\mathcal{E}$. Definition 5.2.2 can be weakened by allowing a negligible error probability, but it is more convenient to work with the stronger condition (and the constructions that we describe later can be made to meet this stronger condition). Some examples of homomorphism relative to interesting circuit families include the following:

Fully homomorphic encryption. We say that $\mathcal{E}$ is *fully homomorphic* if it is correct for a family $\mathcal{C}$ such that $\mathcal{C}_1$ by itself already contains all Boolean circuits. In this case we can ignore the parameter τ in KeyGen; i.e., we can always run it with $\tau = 1$. (Note that the size of the circuit does not figure into the security requirement, and Evaluate always runs in time polynomial in the circuit description, so considering circuits of superpolynomial size in λ does not cause any problems.)

Leveled/somewhat homomorphic encryption. We say that $\mathcal{E}$ is a *leveled* homomorphic encryption scheme if it is correct for a family $\mathcal{C}$ such that, for all τ, $\mathcal{C}_\tau$ contains all Boolean circuits of depth up to τ.

More generally, we may refer to a scheme informally as *somewhat homomorphic* if it is correct for a family $\mathcal{C}$ where the complexity of the circuits in $\mathcal{C}_\tau$ grows with τ. A common example that was used in the first-generation schemes starting with Gentry's original blueprint (e.g., [47, 88, 85, 41]) is when $\mathcal{C}_\tau$ contains circuits computing multivariate polynomials of total degree up to τ and up to 2^τ monomials.

Additively homomorphic encryption. $\mathcal{E}$ is *additively homomorphic* if it is correct for a family $\mathcal{C}$ such that $\mathcal{C}_1$ contains all Boolean circuits made up of only XOR gates. Here too we ignore the parameter τ in KeyGen. One example of additive homomorphism is the Goldwasser–Micali scheme [52].

A weaker version of additive homomorphism (which is realized by most lattice-based encryption schemes) does not support unlimited number of addition operations, but only (say) at most exponential in τ. Namely, $\mathcal{E}$ is *almost additively homomorphic* if it is correct for a family $\mathcal{C}$ where, for all τ, $\mathcal{C}_\tau$ contains all the sums (mod 2) of up to 2^τ variables.

The semantic security of a homomorphic encryption scheme is defined in the usual way [52], without reference to the Evaluate algorithm; indeed Evaluate is a public algorithm with no secrets. Namely, we require that a PPT adversary cannot distinguish between encryptions of 0 and encryptions of 1, even if it knows the public key. Definition 5.2.3 below handles the "functionality parameter" τ by considering it an adversarial quantity (that the adversary must output in unary to ensure that it is polynomially bounded).

Definition 5.2.3 (Semantic security). *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ *be a homomorphic encryption scheme, and let A be an adversary. The* advantage of *A w.r.t.* $\mathcal{E}$ *is defined as*

$$\begin{aligned} Adv_A^{\mathcal{E}}(\lambda) \overset{def}{=} \Bigg| &\Pr\left[A(\mathsf{pk}, c) = 1 : \begin{array}{r} 1^\tau \leftarrow A(1^\lambda),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 1) \end{array}\right] \\ &- \Pr\left[A(\mathsf{pk}, c) = 1 : \begin{array}{r} 1^\tau \leftarrow A(1^\lambda),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0) \end{array}\right]\Bigg| . \end{aligned}$$

The scheme $\mathcal{E}$ *is semantically secure if, for every PPT adversary A, the advantage* $Adv_A^{\mathcal{E}}(\lambda)$ *is negligible in* λ.

In the rest of this tutorial we only consider semantically secure schemes, often without mentioning this requirement.

5.2.1.2 Secret-Key Homomorphic Encryption

It is sometimes convenient to consider secret-key variants of the definitions above, especially since it was shown by Rothblum [82] that these notions are essentially equivalent for homomorphic encryption (see Theorem 5.2.19). The syntax and correctness conditions for secret-key homomorphic encryption are similar to those in Definitions 5.2.1 and 5.2.2 except that KeyGen only outputs the secret key sk rather than the pair $(\mathsf{sk},\mathsf{pk})$, the $\mathsf{Encrypt}$ procedure uses sk rather than pk for encryption, and the $\mathsf{Evaluate}$ procedure gets as input only Π and $\mathbf{c}$ but not pk.

The security definition is also similar, except that, in lieu of the public key, the adversary is given a sequence of ciphertexts (rather than just one), encrypting bits of one of two sequences of its choosing, and it needs to guess which of the two sequences was encrypted.

Definition 5.2.4 (Semantic security, secret-key encryption). *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ *be a secret-key homomorphic encryption scheme, and let A be an adversary. The* advantage *of A w.r.t. $\mathcal{E}$ is defined as*

$$SKAdv_A^{\mathcal{E}}(\lambda) \overset{def}{=} \left| \Pr\left[\begin{array}{l} |\mathbf{b}_0| = |\mathbf{b}_1|\ \& \\ A(\mathsf{pk},\mathbf{c}) = 1 \end{array} : \begin{array}{r} (1^\tau,\mathbf{b}_0,\mathbf{b}_1) \leftarrow A(1^\lambda),\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda,1^\tau), \\ \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{sk},\mathbf{b}_1) \end{array} \right] \right.$$
$$\left. - \Pr\left[\begin{array}{l} |\mathbf{b}_0| = |\mathbf{b}_1|\ \& \\ A(\mathsf{pk},\mathbf{c}) = 1 \end{array} : \begin{array}{r} (1^\tau,\mathbf{b}_0,\mathbf{b}_1) \leftarrow A(1^\lambda),\ \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda,1^\tau), \\ \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{sk},\mathbf{b}_0) \end{array} \right] \right|.$$

The scheme $\mathcal{E}$ is semantically secure if, for every PPT adversary A, the advantage $SKAdv_A^{\mathcal{E}}(\lambda)$ is negligible in λ.

5.2.2 Properties of Homomorphic Encryption Schemes

Property	Description	Defined in	Comments
Secret-key variant	Same key to encrypt, decrypt	Def. 5.2.4	Equivalent to public key variant, Theorem 5.2.19
Strong homomorphism	Evaluated ctxts $\stackrel{\sim}{=}$ fresh ctxts	Defs. 5.2.5,5.2.6	Implies strong compactness, function privacy, and multi-hop
Compactness	Evaluated ctxts are short	Defs. 5.2.8,5.2.9	Strong & weak variants
Function privacy	Evaluated ctxts hide function	Def. 5.2.10	Implied by secure 2PC, Theorem 5.2.13
Multi-hop	Can reprocess evaluated ctxts	Def. 5.2.14	Orthogonal to compactness and function privacy

Table 5.1: Properties of homomorphic encryption

By themselves, the correctness and security properties from above are not enough to rule out uninteresting realizations. Specifically, any secure encryption scheme can be made "homomorphic" by an $\mathsf{Evaluate}$ procedure that simply attaches a description of the circuit Π to the ciphertext tuple $\mathbf{c}$, and a $\mathsf{Decrypt}$ procedure that first decrypts all the ciphertexts and then evaluates Π on the corresponding plaintext bits.

Such uninteresting realizations are ruled out by the additional requirements that we define below. We begin with the simplest and strongest condition, which we call *strong homomorphism*, as well as three weaker (but still useful) conditions, namely *compactness*, *circuit hiding*, and *multi-hop homomorphism*. We also state and prove some lemmas about the relations between these notions. A summary of the definitions and results in this section is given in Table 5.1.

5.2.2.1 Strong Homomorphism

Strong homomorphism requires evaluated ciphertexts to look the same (i.e., have the same distribution) as fresh ciphertexts.

Definition 5.2.5 (Strong homomorphism). *Scheme $\mathcal{E} =$ (KeyGen, Encrypt, Decrypt, Evaluate) is* strongly homomorphic *for a circuit family $\mathcal{C} = \{\mathcal{C}_\tau\}_{\tau\in\mathbb{N}}$ if, for all $\tau \in \mathbb{N}$ and every $\Pi \in \mathcal{C}_\tau$ and plaintext bits $\mathbf{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$, one for every input bit of Π, the two distribution ensembles below are statistically close up to a distance negligible in λ:*

$$\begin{aligned}
\mathsf{Fresh}_{\Pi,\mathbf{b}}(\lambda) &\overset{def}{=} \{(\mathsf{pk}, \mathbf{c}, \mathbf{c}') \ : (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\
&\qquad\qquad \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}),\ \mathbf{c}' \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \Pi(\mathbf{b}))\}, \\
\mathsf{Eval}_{\Pi,\mathbf{b}}(\lambda) &\overset{def}{=} \{(\mathsf{pk}, \mathbf{c}, \mathbf{c}') \ : (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\
&\qquad\qquad \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}),\ \mathbf{c}' \leftarrow \mathsf{Evaluate}(\mathsf{pk}, \Pi, \mathbf{c})\}.
\end{aligned}$$

Definition 5.2.5 can be relaxed to require only computational indistinguishability rather than statistical closeness, but some care must be taken when defining it. Specifically, for some applications we require that even the party that generated the keys cannot distinguish between these two distributions, hence we need them to be indistinguishable *even given the randomness that was used by* KeyGen.

Definition 5.2.6 (Computationally strong homomorphism). *Scheme $\mathcal{E} =$ (KeyGen, Encrypt, Decrypt, Evaluate) is* computationally strongly homomorphic *for a circuit family $\mathcal{C} = \{\mathcal{C}_\tau\}_{\tau\in\mathbb{N}}$ if, for all $\tau \in \mathbb{N}$ and every $\Pi \in \mathcal{C}_\tau$ and plaintext bits $\mathbf{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$, one for every input bit of Π, the two distribution ensembles below are computationally indistinguishable:*

$$\begin{aligned}
\mathsf{Fresh}^*_{\Pi,\mathbf{b}}(\lambda) &\overset{def}{=} \{(r, \mathbf{c}, \mathbf{c}') \ : r \leftarrow \$,\ (\mathsf{sk}, \mathsf{pk}) := \mathsf{KeyGen}(1^\lambda, 1^\tau; r), \\
&\qquad\qquad \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}),\ \mathbf{c}' \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \Pi(\mathbf{b}))\}, \\
\mathsf{Eval}^*_{\Pi,\mathbf{b}}(\lambda) &\overset{def}{=} \{(r, \mathbf{c}, \mathbf{c}') \ : r \leftarrow \$,\ (\mathsf{sk}, \mathsf{pk}) := \mathsf{KeyGen}(1^\lambda, 1^\tau; r), \\
&\qquad\qquad \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}),\ \mathbf{c}' \leftarrow \mathsf{Evaluate}(\mathsf{pk}, \Pi, \mathbf{c})\}.
\end{aligned}$$

Definitions 5.2.5 and 5.2.6 can be modified in the obvious way to handle secret-key homomorphic encryption. Although these definitions are stated relative to a fixed circuit family $\mathcal{C}$, one readily sees that this dependence (as well as the dependence on τ) is immaterial: If some $\mathcal{C}_{\tau^*}$ contains AND and XOR gates (or any other functionally complete set of gates) then we can extend the Evaluate procedure to evaluate any circuit, by repeatedly evaluating each gate on the outputs of the preceding gates. Moreover, the output distribution when evaluating a circuit Π is at most $\mathsf{negl}(\lambda) \cdot \mathsf{size}(\Pi)$ away from that of fresh encryption of the outputs. A similar statement applies also in the computational setting, using the fact that these distributions are indistinguishable even given the secret key (since the correctness condition involved the secret key). Hence we have:

Proposition 5.2.7. *Any encryption scheme which is (computationally) strongly homomorphic relative to a circuit family $\mathcal{C}$, where some $\mathcal{C}_{\tau^*}$ contains AND and XOR gates, can be transformed to a (computationally)* strongly fully homomorphic *scheme (i.e. strongly homomorphic relative to a circuit family $\mathcal{C}'$ with $\mathcal{C}'_1$ containing all circuits).*

Given Proposition 5.2.7, we always assume below that any strongly homomorphic scheme is strongly *fully* homomorphic, and we suppress the irrelevant parameter τ when describing such schemes.

5.2.2.2 Compactness

The main deficiency in the uninteresting realization in which Evaluate just appends the description of Π to the ciphertexts is that we expect the decryption work to be the same whether the decrypted ciphertext is fresh or evaluated. This is clearly the case if the scheme is strongly homomorphic, but being strongly homomorphic is often overkill. A weaker notion that captures a lot of the power of homomorphic computation is *compactness*, which only requires that the *size* of the ciphertext does not grow with the complexity of the evaluated circuit.

Definition 5.2.8 (Compactness). *A homomorphic encryption scheme $\mathcal{E}$ = (KeyGen, Encrypt, Decrypt, Evaluate) is* compact *if there exists a fixed polynomial bound $B(\cdot)$ so that, for all $\lambda, \tau \in \mathbb{N}$, any circuit Π with t inputs and a single output, and plaintext bits $\mathbf{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$, it holds that*

$$\Pr\left[|c'| \le B(\lambda) : \begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}),\ c' \leftarrow \mathsf{Evaluate}(\mathsf{pk}, \Pi, \mathbf{c}) \end{array} \right] = 1.$$

We note that the bound $B(\cdot)$ above depends only on λ but not on τ. This means that, even if we allow some aspects of the scheme (such as the public-key size) to depend on the parameter τ, the size of the evaluated ciphertexts must not grow with τ.

In some settings it is useful to consider a weaker condition, where we allow the ciphertext size to grow with τ so long as it remains smaller then the size of circuits in $\mathcal{C}_\tau$. We use $\log |\mathcal{C}_\tau|$ as our measure of the size of circuits in $\mathcal{C}_\tau$, since you need at least as many bits to describe these circuits (and note that we only count single-output circuits). This yields the following definition:

Definition 5.2.9 (Weak compactness). *A homomorphic encryption scheme $\mathcal{E}$ = (KeyGen, Encrypt, Decrypt, Evaluate) is* weakly compact *if there exists a fixed polynomial bound $B(\cdot,\cdot)$ so that (i) for all $\lambda, \tau \in \mathbb{N}$, any circuit Π with t inputs and a single output, and plaintext bits $\mathbf{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$, it holds that*

$$\Pr\left[|c'| \le B(\lambda,\tau) : \begin{array}{l} (\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk},\mathbf{b}),\ c' \leftarrow \mathsf{Evaluate}(\mathsf{pk},\Pi,\mathbf{c}) \end{array} \right] = 1,$$

and (ii) $B(\lambda,\tau) = poly(\lambda) \cdot o(\log|\mathcal{C}_\tau|)$.

5.2.2.3 Circuit Privacy

Circuit privacy roughly means that the ciphertext generated by Evaluate does not reveal anything about the circuit that it evaluates, beyond the output value of that circuit, even to the party who generated the public and secret keys. To define it, we view the operation of Evaluate as a protocol between a client who generates the keys and encrypts its input, and a server who evaluates some function on that input and returns the result to the client. We then formalize circuit privacy as the usual input privacy property for the server, namely we require that the client can be simulated given only the output value that it learns.

Definition 5.2.10 (Circuit privacy, semi-honest). *A homomorphic encryption scheme* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$*, correct for the circuit family* $\mathcal{C}$*, is circuit private for* $\mathcal{C}$*, if there exists an efficient simulator* Sim *such that, for every* $\tau \in \mathbb{N}$*,* $\Pi \in \mathcal{C}_\tau$*, and plaintext bits* $\mathbf{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$*, one for every input bit of* Π*, we have*

$$\begin{aligned} \mathsf{Real}_{\Pi,\mathbf{b}}(\lambda) &\overset{(c)}{\approx} \mathsf{Sim}(1^\lambda, 1^\tau, \mathbf{b}, \Pi(\mathbf{b})), \textit{ where} \\ \mathsf{Real}_{\Pi,\mathbf{b}}(\lambda) &\overset{def}{=} \{(r, r', \mathbf{c}') : r, r' \leftarrow \$, (\mathsf{sk},\mathsf{pk}) := \mathsf{KeyGen}(1^\lambda, 1^\tau; r), \\ &\qquad \mathbf{c} := \mathsf{Encrypt}(\mathsf{pk},\mathbf{b}; r'),\ \mathbf{c}' \leftarrow \mathsf{Evaluate}(\mathsf{pk},\Pi,\mathbf{c})\}. \end{aligned}$$

It is important to note that the simulator Sim is given the output $\Pi(\mathbf{b})$ *but not the description of* Π *itself*, and it needs to simulate the view that includes the randomness for both key generation and encryption, as well as the evaluated ciphertext.

Circuit privacy against malicious adversaries. Definition 5.2.10 above applies only to the semi-honest case in which the client uses the prescribed KeyGen and Encrypt procedures. The more general case was considered by Ostrovsky et al. [76]. Roughly, they defined circuit privacy against malicious adversaries by requiring that, for every (pk^*, c^*) (even ones that are not generated honestly), there exists an "implied plaintext" b^* such that $\mathsf{Evaluate}(\mathsf{pk}^*, \Pi, c^*)$ can be simulated knowing only pk^*, c^*, and $\Pi(b^*)$ (but without knowing Π itself).

Building on techniques of Gentry et al. [46] (see Theorem 5.2.13 below), they show how to get a compact scheme which is circuit private against malicious adversary, by combining a compact scheme which is not circuit private with a circuit private scheme which is not compact.

5.2.2.4 Circuit Private Homomorphic Encryption Versus Two-Message SFE

As we discussed in the introduction, circuit private homomorphic encryption implies a two-message (semi-honest) secure function evaluation (SFE) protocol for any function. We now show these two notions are essentially equivalent. Namely we show how to realize (non-compact) circuit private fully homomorphic encryption from any two-message semi-honest SFE protocol for general functions.

Recall the structure of a two-message SFE protocol for a function $F(x,y)$, where one party (the client) holds x, the other party (the server) holds y, and the client needs to learn $F(x,y)$.

Definition 5.2.11 (Two-message SFE protocol). *A two-message two-party SFE protocol for a function $F(\cdot,\cdot)$ consists of three procedures, $\mathcal{P}_F = (\mathsf{SFE1}_F, \mathsf{SFE2}_F, \mathsf{SFE3}_F)$ as follows:*

- *The client computes $(s, m_1) \leftarrow \mathsf{SFE1}_F(1^\lambda, x)$, sending m_1 to the server and keeping the state s to itself;*
- *The server responds with $m_2 \leftarrow \mathsf{SFE2}_F(1^\lambda, y, m_1)$;*
- *The client recovers the result as $z \leftarrow \mathsf{SFE3}_F(s, m_2)$.*

The (perfect) correctness of the protocol means that we have $z = F(x,y)$ with probability 1.

Definition 5.2.12 (Semi-honest security). *The security of the protocol $\mathcal{P}_F$ is defined by means of two simulators, Sim_1 and Sim_2, that simulate the view of the two parties. Specifically for all inputs x,y for F, we have*

$$\mathsf{Sim}_1(1^\lambda) \overset{(c)}{\approx} \mathsf{SFE1}_F(1^\lambda, x) \quad \textit{and} \quad \mathsf{Sim}_2(1^\lambda, F(x,y)) \overset{(c)}{\approx} \mathsf{cView}_{x,y}(\lambda), \textit{ where}$$

$$\mathsf{cView}_{x,y}(\lambda) \overset{def}{=} \{(r, m_2) : r \leftarrow \$,\ (s, m_1) := \mathsf{SFE1}_F(1^\lambda, x;\ r),\ m_2 \leftarrow \mathsf{SFE2}_F(1^\lambda, y)\}.$$

Intuitively, a two-message SFE protocol can be thought of as the encryption of x via $\mathsf{SFE1}$ followed by evaluation via $\mathsf{SFE2}$ and decryption via $\mathsf{SFE3}$, but is not quite homomorphic encryption yet. In particular, there is no public key involved, and the same party (the client) is doing both the encryption and the decryption.[4] In contrast, a public key homomorphic encryption should be thought of as a three-player game: first a recipient publishes a public key, then a sender (client) encrypts the data $\mathbf{b}$ under that public key, next an evaluator (server) evaluates a circuit Π on the encrypted data, and finally the recipient decrypts the result and recovers $\Pi(\mathbf{b})$. Another issue is that, as described above, the client's $\mathsf{SFE1}$ function handles the entire input x at once, whereas we need the client processing to be "decomposable" (cf. [61]), i.e., encrypting each bit of x separately.

If the underlying SFE protocol already happened to be decomposable (such as Yao's garbled circuit protocol that uses bit-by-bit oblivious transfer), then it is straightforward to turn it into homomorphic encryption using an auxiliary (standard) public-key encryption scheme. The recipient chooses a public/secret key pair for the encryption scheme, the sender sends the first SFE message and in addition also the encryption of the client's SFE-state s under the public key, and the evaluator forwards the encrypted state to the recipient together with the second SFE message.

[4] We cannot use here the equivalence between public- and secret-key homomorphic encryption from Theorem 5.2.19, since it only applies to *compact schemes*.

The recipient uses its secret key to decrypt and recover the SFE state s, and then uses the procedure $\mathsf{SFE3}$ with this state to recover $F(x, y)$.

Since two-message SFE implies both semantically secure encryption (cf. [49]) and two-message SFE with decomposable client processing (e.g., via Yao's protocol), we get a non-compact circuit private HE from any two-message protocol for secure evaluation of all functions. A more direct proof of this implication, using techniques similar to Gentry's bootstrapping [47], is described in the next theorem.

Theorem 5.2.13. *A circuit private fully homomorphic encryption scheme can be constructed from public-key encryption and a two-message semi-honest SFE protocol for all circuits.*

Proof: Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a public-key encryption scheme, consider the universal function $U(\mathbf{b}, \Pi) = \Pi(\mathbf{b})$, and define the related function $U'(\cdot, \cdot)$ as

$$U'(\mathsf{sk},\ (\mathbf{c}, \Pi)) \stackrel{def}{=} U(\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c}), \Pi)\ \ (= \Pi(\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})).$$

Let $\mathcal{P}_{U'} = (\mathsf{SFE1}_{U'}, \mathsf{SFE2}_{U'}, \mathsf{SFE3}_{U'})$ be a two-message SFE protocol for U', and use $\mathcal{E}$ and $\mathcal{P}_{U'}$ to construct a fully homomorphic public-key encryption scheme $\mathcal{E}' = (\mathsf{KeyGen}', \mathsf{Encrypt}', \mathsf{Decrypt}', \mathsf{Evaluate}')$, as follows:

- $\mathsf{KeyGen}'(1^\lambda)$ runs the underlying key generation $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and then the first-message procedure of the SFE protocol $(s, m_1) \leftarrow \mathsf{SFE1}_{U'}(1^\lambda, \mathsf{sk})$. It outputs $(\mathsf{sk}' = s, \mathsf{pk}' = (\mathsf{pk}, m_1))$ (where pk is used for encryption and m_1 is used for evaluation).
- $\mathsf{Encrypt}'(\mathsf{pk}' = (\mathsf{pk}, m_1), b)$ just uses the $\mathsf{Encrypt}$ procedure of the underlying scheme, outputting $c = \mathsf{Encrypt}(\mathsf{pk}, b)$.
- $\mathsf{Evaluate}'(\mathsf{pk}' = (\mathsf{pk}, m_1),\ \Pi, \mathbf{c})$ runs the second-message procedure of the SFE protocol for U', outputting $\mathbf{c}' \leftarrow \mathsf{SFE2}_{U'}(1^\lambda, m_1,\ (\mathbf{c}, \Pi))$.
- $\mathsf{Decrypt}(\mathsf{sk}' = s, \mathbf{c}')$ runs the last procedure of the SFE protocol for U', outputting $z \leftarrow \mathsf{SFE3}_{U'}(s, c')$.

Correctness of $\mathcal{E}'$ follows from that of $\mathcal{E}$ and $\mathcal{P}_{U'}$: If $\mathbf{c} = \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b})$ then by correctness of $\mathcal{E}$ we have $\mathbf{b} = \mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$, and by correctness of $\mathcal{P}_{U'}$ we have $z = U'(\mathsf{sk},\ (\mathbf{c}, \Pi)) = \Pi(\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})) = \Pi(\mathbf{b})$.

Semantic security follows from the semantic security of the underlying $\mathcal{E}$ and from client security of $\mathcal{P}_{U'}$. To show that $\mathsf{Encrypt}(\mathsf{pk}, 0)$ is indistinguishable from $\mathsf{Encrypt}(\mathsf{pk}, 1)$ even given pk and m_1, consider a hybrid experiment in which m_1 is generated by the simulator $m_1 \leftarrow \mathsf{Sim}_1(1^\lambda)$ instead of being a part of the output of $\mathsf{SFE1}(1^\lambda, \mathsf{sk})$. The client security of $\mathcal{P}_{U'}$ implies that this hybrid experiment is indistinguishable from the real encryption scheme. But in this hybrid, m_1 no longer depends on the secret key sk, and therefore by semantic security of $\mathcal{E}$ we get that $\mathsf{Encrypt}(\mathsf{pk}, 0)$ is indistinguishable from $\mathsf{Encrypt}(\mathsf{pk}, 1)$.

Finally, the circuit privacy of $\mathcal{E}'$ follows from the server security of $\mathcal{P}_{U'}$: The simulator $\mathsf{Sim}_{\mathcal{E}'}$ that we need for circuit privacy is constructed from the client view simulator Sim_2 above: $\mathsf{Sim}_{\mathcal{E}'}(1^\lambda, b, z)$ first chooses randomness r, r' for the key generation and encryption of the underlying encryption scheme, then sets $(\mathsf{sk}, \mathsf{pk}) := \mathsf{KeyGen}(1^\lambda;\ r)$ and $\mathbf{c} := \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b};\ r')$. Next it runs Sim_2 to get $(r'', \mathbf{c}') \leftarrow \mathsf{Sim}_2(1^\lambda, z)$. Then $\mathsf{Sim}_{\mathcal{E}'}$ outputs (r, r'') as the randomness for KeyGen', r' as the randomness for $\mathsf{Encrypt}'$, and $\mathbf{c}'$ as the evaluated ciphertext. Indistinguishability between the simulated and real views follows directly from that of Sim_2. ■

5.2.2.5 Multi-hop Homomorphic Encryption

In settings where we do not have strong homomorphism, the evaluated ciphertexts produced by Evaluate may differ from freshly encrypted ones, bringing up the question of whether one can keep computing on evaluated ciphertexts. An *i-hop* homomorphic encryption scheme is one where Evaluate can be called on its own output up to i times (while still being able to decrypt the result), and a *multi-hop* homomorphic encryption scheme is one which is i-hop for all i. Note that the number of hops supported by a scheme is somewhat orthogonal to its homomorphic capacity. For example the Goldwasser–Micali cryptosystem is only additively homomorphic but is multi-hop, while the scheme constructed from a two-message SFE protocol in Theorem 5.2.13 above is fully homomorphic but supports only a single hop. Also it is clear that strong homomorphism implies multi-hop homomorphism. Gentry et al. studied multi-hop homomorphism in [46]. They extended the notion of circuit privacy to the multi-hop case and described how it can be realized (with or without compactness); their definitions and results are summarized below:

Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ be a homomorphic encryption scheme. The syntax of Evaluate is extended in the natural way to a sequence of circuits: An ordered sequence of circuits $\overrightarrow{\Pi} = (\Pi_1, \ldots, \Pi_t)$ is *compatible* if the output length of Π_j is the same as the input length of Π_{j+1} for all j. The composed function $\Pi_t(\cdots\Pi_2(\Pi_1(\cdot))\cdots)$ is denoted $(\Pi_t \circ \cdots \circ \Pi_1)$. The extended procedure $\mathsf{Evaluate}^*$ takes as input the public key, a compatible sequence $\overrightarrow{\Pi} = (\Pi_1, \ldots, \Pi_t)$, and ciphertexts $\mathbf{c}_0$. For $i = 1, 2, \ldots, t$ it sets $\mathbf{c}_i \leftarrow \mathsf{Evaluate}(\mathsf{pk}, \Pi_i, \mathbf{c}_{i-1})$, outputting the last $\mathbf{c}_t$.

Definition 5.2.14 (Multi-hop homomorphic encryption). *For a circuit family $\mathcal{C}$ and $i \in \mathbb{N}$, we say that $\mathcal{E}$ is i-hop homomorphic for $\mathcal{C}$ if, for all $\lambda, \tau \in \mathbb{N}$ and every compatible sequence $\overrightarrow{\Pi} = (\Pi_1, \ldots, \Pi_t)$ with $t \le i$ functions such that $\tilde{\Pi} = \Pi_t \circ \cdots \circ \Pi_1 \in \mathcal{C}_\tau$, we have*

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c}') = \tilde{\Pi}(\mathbf{b}) : \begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}),\ \mathbf{c}' \leftarrow \mathsf{Evaluate}^*(\mathsf{pk}, \overrightarrow{\Pi}, \mathbf{c}) \end{array}\right] = 1.$$

We say that $\mathcal{E}$ is a multi-hop *homomorphic encryption scheme if it is i-hop for all $i \in \mathbb{N}$.*

Theorem 5.2.15 (Multi-hop homomorphism [46]).

- *If a 1-hop circuit private, fully homomorphic encryption scheme exists, then for any constant i there exists an i-hop circuit private, fully homomorphic encryption scheme.*

- *If both 1-hop circuit private fully homomorphic encryption scheme and 1-hop compact fully homomorphic encryption scheme exist, then there exists a multi-hop circuit private, compact, fully homomorphic encryption scheme.*
- *Under the decision Diffie–Hellman assumption, there exists a (non-compact) multi-hop circuit-private fully homomorphic encryption scheme.*

In the rest of this tutorial we only consider multi-hop schemes.

5.2.2.6 From Secret-Key to Public-Key Homomorphic Encryption

One demonstration of the power of compact homomorphic encryption is a result of Rothblum [82] showing that it enables a transformation from secret-key to public-key encryption. As a warm-up, we demonstrate this result for strongly homomorphic encryption. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ be a *secret-key scheme* which is strongly fully homomorphic. We transform $\mathcal{E}$ to a public-key strongly fully homomorphic scheme $\mathcal{E}' = (\mathsf{KeyGen}', \mathsf{Encrypt}', \mathsf{Decrypt}, \mathsf{Evaluate})$ with the same $\mathsf{Decrypt}$ and $\mathsf{Evaluate}$ procedures, but modified KeyGen' and $\mathsf{Encrypt}'$:

- $\mathsf{KeyGen}'(1^\lambda)$ first runs the underlying KeyGen to get $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$. Then it runs the underlying encryption to encrypt 0 and 1, getting $c_0 \leftarrow \mathsf{Encrypt}(\mathsf{sk}, 0)$ and $c_1 \leftarrow \mathsf{Encrypt}(\mathsf{sk}, 1)$. It outputs $(\mathsf{sk}, \mathsf{pk} = (c_0, c_1))$.
- $\mathsf{Encrypt}'(\mathsf{pk} = (c_0, c_1), b)$ uses the $\mathsf{Evaluate}$ procedure for the underlying scheme. Specifically let Π_{id} be a circuit computing the identity function, then it outputs $c \leftarrow \mathsf{Evaluate}(\Pi_{id}, c_b)$.

The strong homomorphism condition implies that the output of $\mathsf{Encrypt}'$ is statistically close to (resp. computationally indistinguishable from) $\mathsf{Encrypt}(\mathsf{sk}, b)$, even conditioned on the public key (resp. the secret key). Hence the modified scheme maintains the semantic security and strong homomorphism of the underlying scheme, and we have:

Proposition 5.2.16. *Any secret-key strongly homomorphic encryption scheme can be transformed to a public-key strongly homomorphic scheme.*

When the secret-key scheme that we are given is not strongly homomorphic, the transformation above may fail to provide semantic security. Rothblum observed in [82] that semantic security can be obtained by capitalizing on the fact that a compact homomorphic encryption scheme must lose some information in the course of homomorphic evaluation, since the evaluation output is short. Using this loss of information to provide security is done by means of (a special case of) the leftover hash lemma [58], which is stated below.

Lemma 5.2.17 (Linear-hashing extractor). *Fix some $n, q, m \in \mathbb{N}$, and for every matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ consider the multiply-by-$\mathbf{A}$ function $h_{\mathbf{A}}(\mathbf{s}) \overset{def}{=} [\mathbf{A} \times \mathbf{s}]_q$.*

Then for every subset $S \subseteq \mathbb{Z}_q^m$ and every linear subspace $L \subseteq \mathbb{Z}_q^n$, the distribution $\{(\mathbf{A}, h_{\mathbf{A}}(\mathbf{s})) : \mathbf{A} \leftarrow L^m, \mathbf{s} \leftarrow S\}$ is statistically close up to $|L| \cdot \sqrt{2/|S|}$ to the uniform distribution over L^{m+1}.

The special case $q = 2, n = 1$, and $L = \mathbb{Z}_2$ is summarized in the following corollary:

Corollary 5.2.18. *For $m \in \mathbb{N}$ and a bit-string $r \in \{0,1\}^m$, denote the inner-product-with-r function by $h_r(s) \overset{def}{=} [\langle r, s\rangle]_2$. Then for every subset $S \subseteq \{0,1\}^m$, the distribution $\{(r, h_r(s)) : r \leftarrow \{0,1\}^m, s \leftarrow S\}$ is statistically close up to $\sqrt{8/|S|}$ to the uniform distribution over $\{0,1\}^{m+1}$.*

We are now ready to describe the transformation from private-key to a public-key encryption.

Theorem 5.2.19 (Secret-key to public-key [82]). *Any compact, multi-hop,* secret-key *fully homomorphic scheme can be transformed into a compact, multi-hop,* public-key *fully homomorphic scheme.*

Proof: Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ be a compact, multi-hop, *secret-key* fully homomorphic scheme, and let $B(\lambda)$ be a bound on the size of evaluated circuits under $\mathcal{E}$. We transform $\mathcal{E}$ to a public-key scheme $\mathcal{E}' = (\mathsf{KeyGen}', \mathsf{Encrypt}', \mathsf{Decrypt}, \mathsf{Evaluate})$ with the same $\mathsf{Decrypt}$ and $\mathsf{Evaluate}$ procedures, but modified KeyGen' and $\mathsf{Encrypt}'$:

- $\mathsf{KeyGen}'(1^\lambda, 1^\tau)$ first runs the underlying KeyGen to get $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau)$. Letting $m = 2(B(\lambda) + \lambda)$, it next chooses a random string $r \leftarrow \{0,1\}^m$ and for each bit r_i in r it computes $c_i \leftarrow \mathsf{Encrypt}(\mathsf{pk}, r_i)$ and denotes $\mathbf{c}^* = (c_1, \ldots, c_m)$. Finally it outputs the secret key sk and the public key $\mathsf{pk} = (r, \mathbf{c}^*)$.
- $\mathsf{Encrypt}'(\mathsf{pk} = (r, \mathbf{c}), b)$ chooses at random $s \leftarrow \{0,1\}^m$ subject to the condition $\langle r, s\rangle = b \pmod 2$. Denoting by Π_s an m-input circuit made of XOR gates that computes the function $h_s(\cdot)$, it outputs the ciphertext $c \leftarrow \mathsf{Evaluate}(\Pi_s, \mathbf{c}^*)$.

Correctness follows from that of the underlying scheme: Since $\mathbf{c}^* = \mathsf{Encrypt}(\mathsf{sk}, r)$ then for freshly encrypted ciphertexts we have $\mathsf{Decrypt}(\mathsf{sk}, c) = \Pi_s(r) = h_s(r) = b$. Similarly for evaluated ciphertexts, for $\mathbf{c} = \mathsf{Encrypt}'(\mathsf{pk}, \mathbf{b}) = \mathsf{Evaluate}(\Pi_\mathbf{s}, \mathbf{c}^*)$, where each input bit b_i in $\mathbf{b}$ is encrypted using some s_i such that $circuit_{s_i}(r) = b_i$. Then for any Π' such that $\Pi' \circ \Pi_\mathbf{s} \in \mathcal{C}_\tau$, we have

$$\mathsf{Evaluate}(\Pi', \mathbf{c}) = \mathsf{Evaluate}^*(\Pi' \circ \Pi_\mathbf{s}, \mathbf{c}^*) = \Pi'(\Pi_\mathbf{s}(r)) = \Pi'(\mathbf{b}).$$

To prove semantic security, consider a hybrid experiment in which the ciphertexts $\mathbf{c}^*$ in the public key are generated not as encryption of r but rather encryption of 0, $\mathbf{c}^* \leftarrow \mathsf{Encrypt}(\mathsf{sk}, 0)$. By semantic security of the underlying scheme $\mathcal{E}$, no adversary can distinguish between the public key in the real scheme and that in the hybrid experiment, except with negligible probability. It remains to show that, in this hybrid game, the ciphertext is nearly independent of the encrypted bit.

Note that, with this setting for $\mathbf{c}^*$, the evaluated ciphertext $c \leftarrow \mathsf{Evaluate}(\Pi_s, \mathbf{c}^*)$ is independent of the random string r, and it carries at most $B = B(\lambda)$ bits of information about s (since it is only B-bits long). It is therefore sufficient to show that, given $r \in \{0,1\}^m$ and B bits of information about s, one can have at most a negligible advantage in guessing the inner product $b = \mathbf{r}, \mathbf{s} \pmod 2$.

We denote $E_{\mathbf{c}^*}(s) \overset{def}{=} \mathsf{Evaluate}(\Pi_s(\mathbf{c}^*))$, and for any B-bit ciphertext c consider the pre image $E_{\mathbf{c}^*}^{-1}(c) \overset{def}{=} \{s \in \{0,1\}^m : E_{\mathbf{c}^*}(s) = c\}$. By Corollary 5.2.18, for any fixed c, the conditional distribution $D_{\mathbf{c}^*,c} \overset{def}{=} \{(r, \langle r, s\rangle) | E_{\mathbf{c}^*}(s) = c\}$ is statistically close up to $\sqrt{8/\left|E_{\mathbf{c}^*}^{-1}(c)\right|}$ to the uniform distribution over $\{0,1\}^{m+1}$. Hence for every adversary algorithm A, we have

$$\begin{aligned}
&\Pr\left[A(r, E_{\mathbf{c}^*}(s)) = \langle r, s\rangle : r, s \leftarrow \{0,1\}^m\right] \\
&= \sum_{c\in\{0,1\}^B} \Pr[E_{\mathbf{c}^*}(s) = c : s \leftarrow \{0,1\}^m] \cdot \Pr\left[A(r,c) = \langle r, s\rangle : r \leftarrow \{0,1\}^m, s \leftarrow E_{\mathbf{c}^*}^{-1}(c)\right] \\
&\le \sum_{c\in\{0,1\}^B} \frac{\left|E_{\mathbf{c}^*}^{-1}(c)\right|}{2^m} \cdot \left(\underbrace{\Pr\left[A(r,c) = b : r \leftarrow \{0,1\}^m, b \leftarrow \{0,1\}\right]}_{=1/2} + \sqrt{8/\left|E_{\mathbf{c}^*}^{-1}(c)\right|}\right) \\
&= \frac{1}{2} + \sum_{c\in\{0,1\}^B} \frac{\sqrt{8\left|E_{\mathbf{c}^*}^{-1}(c)\right|}}{2^m} < \frac{1}{2} + \frac{2^B \cdot \sqrt{8 \cdot 2^m}}{2^m} < \frac{1}{2} + 3 \cdot 2^{B-m/2} = \frac{1}{2} + 3 \cdot 2^{-\lambda}.
\end{aligned}$$

■

Remark 5.2.20. *For the transformation above, the secret-key scheme $\mathcal{E}$ need not be fully homomorphic (nor multi-hop); for example, it is sufficient for it to be additively homomorphic since the circuits Π_s are all linear. The result $\mathcal{E}'$ would still be a public-key encryption scheme, but it may not be homomorphic since the new encryption procedure used up some of the homomorphic capacity of $\mathcal{E}$. Specifically, if the secret-key scheme $\mathcal{E}$ is i-hop homomorphic relative to some circuit family $\mathcal{C}$, then $\mathcal{E}'$ is $(i-1)$-hop homomorphic relative to the family $\mathcal{C}' = \{\Pi' : (\Pi' \circ \Pi_{\mathbf{s}}) \in \mathcal{C}\ \forall \mathbf{s}\}$.*

This transformation applies also to weakly compact schemes; all we need is for the size of evaluated inner-product ciphertexts to grow slower than the dimension of the vectors used for the inner product.

5.3 Realizing Leveled Homomorphic Encryption

In this section, we show how to implement *leveled* homomorphic encryption, i.e., where the complexity of the encryption scheme grows with the depth of the circuits that it can evaluate. Specifically below we describe the GSW construction due to Gentry, Sahai, and Waters [48] in its most basic form, getting a leveled homomorphic encryption, with semantic security under the (sub exponential) decision-LWE assumption. Later in Section 5.4, we show how to use Gentry's bootstrapping technique to get fully homomorphic encryption (under the additional assumption of circular security), and also quantitatively improve the hardness assumption to quasipolynomial (or even polynomial) decision-LWE.

5.3.1 Tools

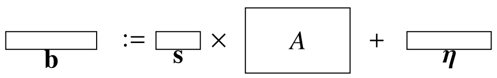
We begin by describing the basic tools that underlie the construction. We describe the learning-with-errors problem, and a *flattening gadget* and some other useful tricks for reducing the norm of vectors.

5.3.1.1 Learning with Errors (LWE)

The learning-with-errors (LWE) problem, first formulated and studied by Regev [80], underlies much of lattice-based cryptography. At a very high level, this average-case computational problem considers noisy modular linear equations, and the hardness assumption states that it is hard to solve such systems (or even decide if a solution exists).

The LWE problem is parametrized by integers q, n, m and a distribution χ over $\mathbb{Z}_q$. The parameter n is related to the security parameter, and we consider a modulus q which is at least polynomially larger than n (or even as large as $q = 2^{n^\epsilon}$) and $m = \Theta(n \log q)$. The distribution χ is concentrated on "small integers", namely we assume $\Pr[x \leftarrow \chi : |x| > \alpha q] < \mathsf{negl}(n)$ for some small $\alpha \ll 1$ (to be determined later).[5] The LWE distribution with these parameters is defined as

$$LWE[n,m,q,\chi] \overset{def}{=} \{(\mathbf{A},\mathbf{b}) \;:\; \mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m},\ \mathbf{s} \leftarrow \mathbb{Z}_q^n,\ \boldsymbol{\eta} \leftarrow \chi^m,\ \mathbf{b} := [\mathbf{s}\mathbf{A} + \boldsymbol{\eta}]_q\}.$$



In words, we choose a uniformly random n-by-m matrix $\mathbf{A}$ and a row n-vector $\mathbf{s}$ over $\mathbb{Z}_q$, and an m-vector $\boldsymbol{\eta}$ whose entries are drawn from χ, and compute $\mathbf{b} := [\mathbf{s}\mathbf{A} + \boldsymbol{\eta}]_q$. The vector $\mathbf{s}$ is called *the secret* and $\boldsymbol{\eta}$ is called *the noise* (or the error).

Remark 5.3.1. *Applebaum et al. proved in [4] that a variant in which* $\mathbf{s}$ *is drawn from* χ^n *(rather than being uniformly random) is equally hard. Hence we formalize the hardness assumption below using a uniform secret* $\mathbf{s}$*, but we can use it with a small secret when needed. For most of this tutorial we get by with a uniform* $\mathbf{s}$*, but on occasion also consider variants that need the secret to be small.*

Since m is significantly larger than n, the row-span of $\mathbf{A}$ is a rather low-dimension random linear subspace of $\mathbb{Z}_q^m$. It follows that with high probability, the row span of $\mathbf{A}$ has large minimum distance (in l_∞ norm), roughly $q^{1-n/m}$. Hence for $\alpha \ll q^{-n/m}$, it holds with high probability (over $\mathcal{A}$) that there is a unique point in the row span of $\mathbf{A}$ within distance αq of $\mathbf{b}$ (call that point $\mathbf{b}'$), and the secret $\mathbf{s}$ is the unique solution to $\mathbf{s}\mathbf{A} = \mathbf{b}' \pmod{q}$.

The argument above shows that with high probability the secret $\mathbf{s}$ is uniquely defined, but computing it from $\mathbf{A}$ and $\mathbf{b}$ seems to be hard. Indeed, Regev described in [80] a worst-case to average-case quantum reduction from approximating the

[5] Often χ is taken to be a discrete Gaussian distribution with parameter $\alpha' \approx \alpha$ [71].

shortest-vector search problem in an arbitrary dimension-n lattice to solving a random instance of LWE, where the approximation factor is essentially n/α. Follow-up work [78, 18] described classical reductions of some other worst-case problems to LWE with similar dependence on α. These reductions provide ample evidence of the hardness of the search problem of computing $\mathbf{s}$ from $\mathbf{A}$ and $\mathbf{b}$. Moreover, for many parameter regimes, the search problem of computing $\mathbf{s}$ can be further reduced to the problem of distinguishing the pair $(\mathbf{A}, \mathbf{b})$ from uniform [8, 80, 70]. It is the hardness of this decision problem which is most convenient for use in cryptography.

We often think of the pair $(\mathbf{A}, \mathbf{b})$ as an $(n + 1)$-by-m matrix with $\mathbf{b}$ being the last row. Denoting this matrix by $\mathbf{A}'$, it satisfies the equation $(\mathbf{s}, -1) \times \mathbf{A}' = \boldsymbol{\eta} \pmod{q}$ with $\boldsymbol{\eta}$ having low norm, yet our hardness assumption says that $\mathbf{A}'$ is pseudorandom in $\mathbb{Z}^{(n+1)\times m}$. Thinking of $n' = n + 1$ as the security parameter, we can therefore state our hardness assumption as follows:

Definition 5.3.2 (Decision-LWE). *For parameters n, m, q, α (that depend on the security parameter λ), the decision-LWE hardness assumption (DLWE$[n, m, q, \alpha]$) states that there exists an efficiently sampleable ensemble $\{\psi_\lambda\}_\lambda$ over pairs $(\mathbf{s} \in \mathbb{Z}_q^n, \mathbf{A} \in \mathbb{Z}_q^{n\times m})$, for which the following conditions hold:*

- *The induced distribution ensemble over $\mathbf{A}$ is pseudorandom over $\mathbb{Z}_q^{n\times m}$.*
- *The l_∞-norm of $\boldsymbol{\eta} \overset{def}{=} [\mathbf{s}\mathbf{A}]_q \in \mathbb{Z}_q^m$ is bounded by αq with overwhelming probability, and $s_n = -1$.*

(Note that above we use the notations $n, \mathbf{s}, A$ rather than $n', \mathbf{s}', \mathbf{A}'$ from before; this will be more convenient in the sequel.)

Remark 5.3.3. *The DLWE assumption becomes stronger as α gets smaller. We have strong evidence that it holds for any polynomial fraction $\alpha(n) = 1/poly(n)$, and distinguishing $\mathbf{A}$ from random seems hard also for quasipolynomial or even nearly exponential fractions such as 2^{-n^ϵ}. On the other hand, for an exponentially small fraction, $\alpha = 2^{-O(n)}$, lattice-reduction tools allow us to find the secret $\mathbf{s}$ in polynomial time (so in particular we can distinguish $\mathbf{A}$ from random).*

Also we clearly need $\alpha q \geq 1$ (else the only integer vector satisfying $\|\boldsymbol{\eta}\|_\infty \leq \alpha q$ is the all-zero vector), and there are attacks due to Arora and Ge [5] that apply when $\alpha q = O(1)$. Below we always use settings where $\alpha q = n$ (which is nearly the smallest possible). Hence in our setting the DLWE hardness assumption becomes stronger as q increases, and for $q = 2^{\Omega(n)}$ we know that it no longer holds. In this section we will need to assume that DLWE holds for $q = 2^{n^\epsilon}$, but later in the text we can relax this to $q = 2^{polylog(n)}$ and even $q = poly(n)$.

Finally note that the assumption becomes stronger as m increases, but all evidence points to this assumption holding for every polynomial $m = m(n)$. Below we will use $m \approx 2n \log q$.

5.3.1.2 Public-Key Encryption from LWE

Regev described in [80] a simple public-key encryption whose security is based on the DLWE assumption. That construction, which is naturally additively homomorphic, plays an important role in many homomorphic encryption schemes, including the GSW scheme that we describe later in this section.

The salient features of the Regev encryption scheme is that the secret key is a vector $\mathbf{s} \in \mathbb{Z}_q^n$ with $s_n = -1$, and an encryption of a bit b is a vector $\mathbf{u} \in \mathbb{Z}_q^n$ such that $\langle \mathbf{s}, \mathbf{u} \rangle = b \cdot \lfloor q/2 \rfloor + \delta \pmod q$ where δ is a small noise. Clearly adding/subtracting two ciphertexts yields an encryption of the XOR of the two encrypted bits, with noise which is the sum of the two noise elements for the individual ciphertext. It is easy to see how to construct a secret-key encryption scheme with ciphertexts as above, and since that scheme is additively homomorphic then one can use Theorem 5.2.19 to turn it into a public-key scheme. A more direct way of getting a public-key scheme is described below. Security of this scheme is reduced to the DLWE assumption; the proof is nearly identical to Lemma 5.3.6 later in this section.

Key generation: $\mathsf{KeyGen}(\mathbf{1}^\lambda)$**.** given the parameters n, m, q and the distribution ψ_n for the DLWE assumption, draw a pair $(\mathbf{s}, \mathbf{A}) \leftarrow \psi_n$, outputting the secret key $\mathbf{s} \in \mathbb{Z}_q^n$ and public key $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$.

Encryption: $\mathsf{Encrypt}(\mathbf{A}, \mathbf{b})$**.** For a public key $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and plaintext bit $b \in \{0, 1\}$, choose a uniform $\{0, 1\}$ vector $\mathbf{r} \leftarrow \{0, 1\}^m$ and output the ciphertext vector $\mathbf{u} := [b \cdot \lfloor q/2 \rfloor \cdot (0, \ldots, 0, -1) + \mathbf{A} \times \mathbf{r}]_q \in \mathbb{Z}_q^n$.

Observe that indeed $\langle \mathbf{s}, \mathbf{u} \rangle = [b \cdot \lfloor q/2 \rfloor + \mathbf{sAr} = b \cdot \lfloor q/2 \rfloor + \langle \boldsymbol{\eta}, \mathbf{r} \rangle$, and $|\langle \boldsymbol{\eta}, \mathbf{r} \rangle|$ is small as needed.

Decryption: $\mathsf{Decrypt}(\mathbf{s}, \mathbf{u})$**.** Compute $z := [\langle \mathbf{s}, \mathbf{u} \rangle]_q$, outputting 0 if $|z| < q/4$ and 1 otherwise.

5.3.1.3 The Flattening Gadget

A very useful technical tool in many lattice-based cryptosystems (including GSW encryption) is a "flattening gadget" that allows one to take a high-norm vector and represent it by a low-norm vector of higher dimension, while maintaining some linear-algebraic properties. We want a "representation function" $f : \mathbb{Z}_q \to \mathbb{Z}_q^\ell$ (for some not-too-large ℓ) with the properties:

- For any $z \in \mathbb{Z}_q$, the l_∞-norm of $f(z)$ is much smaller than q; and
- Recovering z from $f(z)$ is a linear operation. That is, there exists a "gadget vector" $\mathbf{g} \in \mathbb{Z}_q^\ell$ such that for every $z \in \mathbb{Z}_q$ we have $\langle \mathbf{g}, f(z) \rangle = z \pmod q$.

Note that the function f itself need not be linear, but its inverse is linear. A simple function with these properties is obtained by breaking z into its binary representation: Let $\ell = \lceil \log q \rceil$ and $f(z) = (z_0, z_1, \ldots, z_{\ell-1})$ be the vector of bits in the binary (2's-complement) representation of $z \in [-q/2, q/2)$, with z_0 the least-significant bit and $z_{\ell-1} \in \{0, -1\}$ representing the most-significant bit. Then $\|f(z)\|_\infty \leq 1$ and setting $\mathbf{g} = (1, 2, \ldots, 2^{\ell-1})$ we have $\langle \mathbf{g}, f(z) \rangle = \sum_{i=1}^{\ell-1} 2^i z_i = z$.

To stress the fact that our representation function is an inverse of a linear function, we denote it below by $g^{-1}(\cdot)$. Note again that $\mathbf{g}$ is a vector (representing a linear function), but g^{-1} is a non-linear function.

The same representation extends naturally to vectors and matrices. For an n-vector $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{Z}^n$, we let $G^{-1}(\mathbf{z})$ be the concatenation of all the vectors $g^{-1}(z_i)$, namely $G^{-1}(\mathbf{z}) \overset{def}{=} (g^{-1}(z_1)|g^{-1}(z_2)|\ldots|g^{-1}(z_n)) \in \mathbb{Z}_q^{n\ell}$. If we think of $\mathbf{z}$ and $G^{-1}(\mathbf{z})$ as column vectors and consider the "gadget matrix"

$$\mathbf{G} \overset{def}{=} \begin{pmatrix} 1\ 2 \ldots 2^{\ell-1} & & & \\ & 1\ 2 \ldots 2^{\ell-1} & & \\ & & \ddots & \\ & & & 1\ 2 \ldots 2^{\ell-1} \end{pmatrix} \in \mathbb{Z}_q^{n\times n\ell},$$

then for any vector $\mathbf{z} \in \mathbb{Z}_q^n$ we have $\|G^{-1}(\mathbf{z})\|_\infty \le 1$ and $\mathbf{G} \times G^{-1}(\mathbf{z}) = \mathbf{z}$. Similarly for an $n \times m$ matrix $\mathbf{A} = (\mathbf{a}_1|\ldots|\mathbf{a}_m)$ with columns $\mathbf{a}_i \in \mathbb{Z}_q^n$, we let

$$G^{-1}(\mathbf{A}) \overset{def}{=} (G^{-1}(\mathbf{a}_1)|\ldots|G^{-1}(\mathbf{a}_m)) \in \mathbb{Z}_q^{n\ell\times m}.$$

Then for any matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ we have $\|G^{-1}(\mathbf{A})\|_\infty \le 1$ and $\mathbf{G} \times G^{-1}(\mathbf{A}) = \mathbf{A}$.

5.3.1.4 Modulus and Key Switching

Below we describe the two tricks of modulus switching and key switching, due to Brakerski et al. [19, 17], which can be used in some cases to reduce the size of the modulus and the dimension of the LWE secret vectors. These tricks are not strictly needed to obtain fully homomorphic encryption, but they can be used to improve its parameters and efficiency and to quantitatively weaken the hardness assumptions that are needed for its security.

Modulus switching. Modulus switching lets one convert approximate linear relations modulo one integer into relations modulo another.

Lemma 5.3.4. *Fix the dimension $n \in \mathbb{N}$ and moduli $p, q \in \mathbb{N}$, and let $\mathbf{s}, \mathbf{u} \in \mathbb{Z}_q^n$, such that their mod-$q$ inner product is bounded away from $q/4$, specifically $[\langle \mathbf{s}, \mathbf{u}\rangle]_q = b \cdot \lfloor q/2 \rfloor + \delta$ where $|\delta| < q(\frac{1}{4} - \frac{\|\mathbf{s}\|_1}{2p})$.*

Let $\mathbf{u}' = \lceil \mathbf{u} \cdot p/q \rfloor$, then $[\langle \mathbf{s}, \mathbf{u}\rangle]_p = b \cdot \lfloor p/2 \rfloor + \delta'$ where $|\delta'| \le \frac{p}{q}|\delta| + \frac{\|\mathbf{s}\|_1}{2} < p/4$.

Proof: Let $\boldsymbol{\epsilon}$ denote the rounding error in the computation of $\mathbf{u}'$, i.e., $\mathbf{u}' = p/q \cdot \mathbf{u} + \boldsymbol{\epsilon}$, note that $\|\boldsymbol{\epsilon}\|_\infty \le 1/2$. Since over the integers we have the equality $\langle \mathbf{s}, \mathbf{u}\rangle = b \cdot \lfloor q/2 \rfloor + \delta + kq$ for some $k \in \mathbb{N}$, then we get

$$\langle \mathbf{s}, \mathbf{u}'\rangle = (p/q)\langle \mathbf{s}, \mathbf{u}\rangle + \langle \mathbf{s}, \boldsymbol{\epsilon}\rangle = b \cdot \lfloor p/2 \rfloor + \underbrace{\frac{p}{q}\delta + \langle \mathbf{s}, \boldsymbol{\epsilon}\rangle}_{=\delta'} + kp,$$

with $|\delta'| < \frac{p}{q}|\delta| + \frac{\|\mathbf{s}\|_1}{2} < q/4$, as needed. ■

We can use this lemma in settings where we have a low-norm secret LWE vector $\mathbf{s}$ (such as when it is drawn from the error distribution χ, as described by Applebaum et al. [4]). In this situation we can switch from a mod-q vector $\mathbf{u}$ to a mod-p vector $\mathbf{u}'$ for $p \ll q$, with only a small penalty in terms of increased added noise. As $p \ll q$, then mod-p arithmetic has smaller complexity than mod-q arithmetic.

Key switching. Key switching let us publish a public pseudorandom gadget for converting approximate linear relations relative to a secret vector $\mathbf{t}$ into relations relative to another vector $\mathbf{s}$.

Specifically, fix the DLWE parameters n, m, q, α, where $m = n' \lceil \log q \rceil$ for some n'. Let $(\mathbf{s}, \mathbf{A}) \leftarrow \psi$ be drawn according to the distribution from Definition 5.3.2, so $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is pseudorandom, $s_n = -1$, and $\boldsymbol{\eta} = [\mathbf{sA}]_q$ has low norm $\|\boldsymbol{\eta}\|_\infty < \alpha q$.

Fix any vector $\mathbf{t} \in \mathbb{Z}_q^{n'}$ and let $\mathbf{A}'_{\mathbf{s}:\mathbf{t}} \in \mathbb{Z}_q^{n\times m}$ be the matrix obtained by subtracting $\mathbf{tG}$ modulo q from the last row of $\mathbf{A}$, where $\mathbf{G} \in \mathbb{Z}_q^{n'\times m}$ is the matrix from the flattening gadget above. (Recall that $m = n' \lceil \log q \rceil$, as needed for this gadget.) Clearly, since $\mathbf{A}$ is pseudorandom then so is $\mathbf{A}'_{\mathbf{s}:\mathbf{t}}$ for any fixed $\mathbf{t}$ independent of $\mathbf{s}$. Also since $s_n = -1$ then

$$\mathbf{sA}'_{\mathbf{s}:\mathbf{t}} \;=\; \mathbf{sA} + \mathbf{tG} \;=\; \mathbf{tG} + \boldsymbol{\eta} \pmod{q}.$$

The use of the key-switching gadget $\mathbf{A}'_{\mathbf{s}:\mathbf{t}}$ is summarized in the following lemma:

Lemma 5.3.5. *Fix the parameters $n', n, m, q \in \mathbb{N}$ as above, and let $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{t} \in \mathbb{Z}_q^{n'}$, and $\mathbf{A}'_{\mathbf{s}:\mathbf{t}} \in \mathbb{Z}_q^{n\times m}$ such that $\mathbf{sA}'_{\mathbf{s}:\mathbf{t}} = \mathbf{tG} + \boldsymbol{\eta} \pmod{q}$.*

Let $\mathbf{u}' \in \mathbb{Z}_q^{n'}$ be a vector whose mod-q inner product with $\mathbf{t}$ is bounded away from $q/4$, specifically $[\langle \mathbf{t}, \mathbf{u}' \rangle]_q = b \cdot \lfloor q/2 \rfloor + \delta'$ where $|\delta'| < q/4 - \|\boldsymbol{\eta}\|_1$. Computing $\mathbf{u} := [\mathbf{A}'_{\mathbf{s}:\mathbf{t}} \times G^{-1}(\mathbf{u}')]_q$, we have

$$[\langle \mathbf{s}, \mathbf{u} \rangle]_p = b \cdot \lfloor q/2 \rfloor + \delta,$$

where $|\delta| \le |\delta'| + \|\boldsymbol{\eta}\|_1 < p/4$.

Proof:

$$\begin{aligned}\langle \mathbf{s}, \mathbf{u} \rangle &= \mathbf{s} \times \mathbf{A}'_{\mathbf{s}:\mathbf{t}} \times G^{-1}(\mathbf{u}') = (\mathbf{tG} + \boldsymbol{\eta}) \times G^{-1}(\mathbf{u}') = \mathbf{tG} \times G^{-1}(\mathbf{u}') + \left\langle \boldsymbol{\eta}, G^{-1}(\mathbf{u}') \right\rangle \\ &= \langle \mathbf{t}, \mathbf{u}' \rangle + \left\langle \boldsymbol{\eta}, G^{-1}(\mathbf{u}') \right\rangle = b \cdot \lfloor q/2 \rfloor + \underbrace{\delta' + \left\langle \boldsymbol{\eta}, G^{-1}(\mathbf{u}') \right\rangle}_{=\delta},\end{aligned}$$

and since $G^{-1}(\mathbf{u}')$ is a dimension-m 0/1 vector then $|\left\langle \boldsymbol{\eta}, G^{-1}(\mathbf{u}') \right\rangle| < \|\boldsymbol{\eta}\|_1$, hence $|\delta| \le |\delta'| + \|\boldsymbol{\eta}\|_1 < p/4$. ■

One use of Lemma 5.3.5 is to reduce the dimension of an LWE secret from n' to $n < n'$, which we can do as long as n is large enough so that the $DLWE[n, m, q, \alpha]$ hardness assumption still holds. As with modulus-switching, this can be used to get lower-complexity arithmetic operations.

5.3.2 The GSW Encryption Scheme

5.3.2.1 First Try

The high-level intuition for the GSW scheme [48] comes from the concepts of *eigenvectors and eigenvalues* in linear algebra. Let $\mathbb{F}$ be some field and let $\mathbf{C} \in \mathbb{F}^{n\times n}$ be a square matrix over $\mathbb{F}$. Recall that $\mathbf{s} \in \mathbb{F}^n$ is a (left-) eigenvector of $\mathbf{C}$ with corresponding eigenvalue $u \in \mathbb{F}$ if we have $\mathbf{s} \times \mathbf{C} = \mathbf{s} \cdot u$ (with operations in $\mathbb{F}$). It is easy to see that, if $\mathbf{C}_1, \mathbf{C}_2$ are two n-by-n matrices that share the same eigenvector $\mathbf{s}$ with corresponding eigenvalues u_1, u_2, then

$$\mathbf{s} \times (\mathbf{C}_1 \pm \mathbf{C}_2) \;=\; \mathbf{s} \cdot (u_1 \pm u_2) \;\text{ and }\; \mathbf{s} \times (\mathbf{C}_1 \times \mathbf{C}_2) \;=\; \mathbf{s} \cdot (u_1 \cdot u_2).$$

In words, the scalar $u_1 \pm u_2$ is the eigenvalue of $\mathbf{C}_1 \pm \mathbf{C}_2$ corresponding to the eigenvector $\mathbf{s}$, and similarly $u_1 \cdot u_2$ is the eigenvalue of $\mathbf{C}_1 \times \mathbf{C}_2$ corresponding to $\mathbf{s}$.

It is therefore tempting to construct a cryptosystem where $\mathbf{s}$ is the secret key and an encryption of a value u is a matrix that has $\mathbf{s}$ as an eigenvector with u the corresponding eigenvalue. Then we could use matrix addition and multiplication to implement homomorphic addition and multiplication over $\mathbb{F}$, getting a fully homomorphic scheme. The problem, of course, is that this scheme is insecure: Given the ciphertext matrix $\mathbf{C}$ it is easy to compute the eigenvectors of $\mathbf{C}$, and one of these eigenvectors is the secret key.

5.3.2.2 Second Try

Attempting to improve security, we may try adding some noise, making $\mathbf{s}$ an *approximate* rather than an exact eigenvector of the ciphertext matrices, and relying on the hardness of LWE to get security. Specifically, we would like to work over $\mathbb{Z}_q$ and maintain the invariant that a plaintext scalar u is encrypted relative to a secret key $\mathbf{s}$ by a matrix $\mathbf{C}$ such that $\mathbf{s} \times \mathbf{C} \approx \mathbf{s} \cdot u = \mathbf{s} \times \mathbf{C} + \boldsymbol{\eta}$ for a small noise vector $\boldsymbol{\eta}$. We note that adding noise over the real field would have very little effect, since the algorithm for computing eigenvectors over the reals is very geometric in spirit, and is robust to inaccuracies or noise. On the other hand, computing eigenvectors over discrete fields is done using Gaussian elimination, which is algebraic and very brittle in the presence of small noise.

Adding some noise may help security, but does it hurt the homomorphic operations? At least for addition things still seem to work: If we have $\mathbf{s}\times\mathbf{C}_i = \mathbf{s}\cdot u_i + \boldsymbol{\eta}_i$ for $i = 1, 2$ then also $\mathbf{s} \times (\mathbf{C}_1 \pm \mathbf{C}_2) = \mathbf{s} \cdot (u_1 \pm u_2) + (\boldsymbol{\eta}_1 \pm \boldsymbol{\eta}_2)$ so $\mathbf{s}$ is still an approximate eigenvector of the matrix $(\mathbf{C}_1 \pm \mathbf{C}_2)$, corresponding to the eigenvalue $(u_1 \pm u_2)$ with the (still small) noise vector $(\boldsymbol{\eta}_1 \pm \boldsymbol{\eta}_2)$. For multiplication, however, we have

$$\begin{aligned}\mathbf{s} \times (\mathbf{C}_1 \times \mathbf{C}_2) &= (\mathbf{s} \cdot u_1 + \boldsymbol{\eta}_1) \times \mathbf{C}_2 \;=\; \mathbf{s} \times \mathbf{C}_2 \cdot u_1 + \boldsymbol{\eta}_1 \times \mathbf{C}_2 \\ &= (\mathbf{s} \cdot u_2 + \boldsymbol{\eta}_2) \cdot u_1 + \boldsymbol{\eta}_1 \times \mathbf{C}_2 \\ &= \mathbf{s} \cdot u_1 u_2 + (u_1 \cdot \boldsymbol{\eta}_2 + \boldsymbol{\eta}_1 \times \mathbf{C}_2) \pmod{q}. \qquad (5.1)\end{aligned}$$

We would like to think of $\mathbf{s}$ as an approximate eigenvector of $\mathbf{C}_1 \times \mathbf{C}_2$ corresponding to the eigenvalue $(u_1 \cdot u_2)$ with noise vector $\boldsymbol{\eta}^* = (u_1 \cdot \boldsymbol{\eta}_2 + \boldsymbol{\eta}_1 \times \mathbf{C}_2)$, but $\boldsymbol{\eta}^*$ may not be small anymore: If the ciphertext matrix $\mathbf{C}_2$ has large entries, then so would the vector $\boldsymbol{\eta}^*$, no matter how small $\boldsymbol{\eta}_1$ is. Similarly, if the plaintext scalar u_1 is large then $u_1 \cdot \boldsymbol{\eta}_2$ will be large.

5.3.2.3 Final Try

To recover functionality, we want to ensure that both the plaintext scalars and ciphertext matrices are kept small. Keeping the plaintext small can be done by encrypting only 0's and 1's and using only NAND gates, which are implemented over $\mathbb{Z}_q$ as $NAND(x, y) = 1 - xy$.

To ensure that the ciphertext matrices have small entries we use the flattening gadget from Section 5.3.1.3. Recalling the form of the noise from Equation (5.1), we would like to use the low-norm $G^{-1}(\mathbf{C}_2)$ instead of $\mathbf{C}_2$ itself in the multiplication procedure. Namely, we want the homomorphic multiplication procedure to be $\mathbf{C}_1 \times G^{-1}(\mathbf{C}_2)$. To make the dimensions match, we need the ciphertext matrices $\mathbf{C}$ to be n-by-N matrices with $N = n\ell$ (recall that $\ell = \lceil \log q \rceil$).

To maintain correctness we need to introduce the gadget $\mathbf{G}$ into the invariant that we maintain: A plaintext scalar u is now encrypted relative to a secret key $\mathbf{s}$ by a matrix $\mathbf{C}$ such that $\mathbf{s} \times \mathbf{C} \approx \mathbf{s}' \cdot u$, where $\mathbf{s}' = \mathbf{s} \times \mathbf{G}$. With this modification, let $\mathbf{C}_i \in \mathbb{Z}_q^{n \times N}$ (for $i = 1, 2$) be two matrices that encrypt two bits $b_i \in \{0, 1\}$ relative to the secret key $\mathbf{s}$, i.e., $\mathbf{s} \times \mathbf{C}_i = b_i \cdot (\mathbf{s} \times \mathbf{G}) + \boldsymbol{\eta}_i$ for small noise vectors $\boldsymbol{\eta}_i$. Then we have

$$\begin{aligned}
\mathbf{s} \times (\mathbf{C}_1 \times G^{-1}(\mathbf{C}_2)) &= (b_1 \cdot \mathbf{s} \times \mathbf{G} + \boldsymbol{\eta}_1) \times G^{-1}(\mathbf{C}_2) \\
&= b_1 \cdot \mathbf{s} \times \mathbf{G} \times G^{-1}(\mathbf{C}_2) + \boldsymbol{\eta}_1 \times G^{-1}(\mathbf{C}_2) \\
&= b_1 \cdot \mathbf{s} \times \mathbf{C}_2 + \boldsymbol{\eta}_1 \times G^{-1}(\mathbf{C}_2) = b_1(b_2 \cdot \mathbf{s} \times \mathbf{G} + \boldsymbol{\eta}_2) + \boldsymbol{\eta}_1 \times G^{-1}(\mathbf{C}_2) \\
&= b_1 b_2 \cdot (\mathbf{s} \times \mathbf{G}) + (b_1 \cdot \boldsymbol{\eta}_2 + \boldsymbol{\eta}_1 \times G^{-1}(\mathbf{C}_2)) \pmod{q}
\end{aligned}$$

Let us define the homomorphic NAND operation as

$$\mathsf{homNAND}(\mathbf{C}_1, \mathbf{C}_2) \overset{def}{=} [\mathbf{G} - (\mathbf{C}_1 \times G^{-1}(\mathbf{C}_2))]_q, \tag{5.2}$$

so we get

$$\begin{aligned}
\mathbf{s} \times \mathsf{homNAND}(\mathbf{C}_1, \mathbf{C}_2) &= \mathbf{s} \times \mathbf{G} - \mathbf{s} \times \mathbf{C}_1 \times G^{-1}(\mathbf{C}_2) &(5.3)\\
&= \underbrace{(1 - b_1 b_2)}_{NAND(b_1, b_2)} \cdot (\mathbf{s} \times \mathbf{G}) - \underbrace{(b_1 \cdot \boldsymbol{\eta}_2 + \boldsymbol{\eta}_1 \times G^{-1}(\mathbf{C}_2))}_{\boldsymbol{\eta}'} \pmod{q}.
\end{aligned}$$

As $b_1 \in \{0, 1\}$ and $G^{-1}(\mathbf{C}_2) \in \{-1, 0, 1\}^{N \times N}$, the l_∞ norm of $\boldsymbol{\eta}'$ is bounded by

$$\|\boldsymbol{\eta}'\|_\infty \le \|\boldsymbol{\eta}_2\|_\infty + N \cdot \|\boldsymbol{\eta}_1\|_\infty \le (N + 1) \cdot \max\{\|\boldsymbol{\eta}_1\|, \|\boldsymbol{\eta}_2\|\}.$$

Consider now a depth-d circuit made of NAND gates, and assume that the inputs to the circuit are encrypted with noise vectors with norm below some bound B (below we use $B = n \cdot m$). Then the noise vectors at level i of the circuit have norm bounded by $B \cdot (N+1)^i$, and in particular the output has noise bounded by $B \cdot (N+1)^d$.

5.3.2.4 The GSW Leveled Scheme

We are now ready to describe the complete construction. We already established that the secret key is a vector $\mathbf{s} \in \mathbb{Z}_q^n$ and ciphertexts are matrices $\mathbf{C} \in \mathbb{Z}_q^{n\times N}$, and that we maintain the invariant that an encryption of a bit $b \in \{0,1\}$ at level i of the circuit satisfies $\mathbf{s} \times \mathbf{C} = b \cdot (\mathbf{s} \times \mathbf{G}) + \boldsymbol{\eta}$ where $||\boldsymbol{\eta}||_\infty \le n \cdot m \cdot (N+1)^i$. It remains to explain how to encrypt and decrypt, set the parameters, and argue security.

<u>KeyGen($\mathbf{1}^\lambda, \mathbf{1}^\tau$).</u> We first select the DLWE parameters n, m, q, ψ_n (with $\alpha q = n$) so as to enable homomorphic evaluation of circuits of depth up to τ. This is described later in this section. Denoting $N = n \cdot \lceil \log q \rceil$, we remark that the parameters are chosen so that $n \cdot m \cdot (N+1)^{\tau+1} < q/4$.

The KeyGen procedure draws a pair $(\mathbf{s}, \mathbf{A}) \leftarrow \psi_n$, outputting the secret key $\mathbf{s} \in \mathbb{Z}_q^n$ and public key $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, after ensuring that $\mathbf{s} \neq 0$ and that the l_∞-norm of $\boldsymbol{\eta} = [\mathbf{sA}]_q$ is bounded by n (else it draws another pair).

<u>Encrypt($\mathbf{A}, b$).</u> For a public key $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and plaintext bit $b \in \{0,1\}$, the Encrypt procedure chooses a uniform $\{0,1\}$ matrix $\mathbf{R} \leftarrow \{0,1\}^{m\times N}$ and outputs the ciphertext matrix $\mathbf{C} := [b \cdot \mathbf{G} + \mathbf{A} \times \mathbf{R}]_q \in \mathbb{Z}_q^{n\times N}$.

We observe that $\mathbf{sC} = b \cdot \mathbf{sG} + \mathbf{sAR} = b \cdot \mathbf{sG} + \boldsymbol{\eta}\mathbf{R}$, and $||\boldsymbol{\eta}\mathbf{R}||_\infty \le n \cdot m$, so $\mathbf{C}$ satisfies our invariant for fresh ciphertexts.

<u>Evaluate($\Pi, \overrightarrow{\mathbf{C}}$).</u> For a circuit of NAND gates of depth up to τ, go over the circuit in topological order from inputs to outputs; for every gate with inputs encrypted by $\mathbf{C}_1, \mathbf{C}_2$, compute the output encryption as $\mathbf{C}_{out} := \mathbf{G} - \mathbf{C}_1 \times G^{-1}(\mathbf{C}_2)$.

By the analysis from above, a ciphertext $\mathbf{C}$ at level i of the circuit satisfies $\mathbf{sC} = b \cdot \mathbf{sG} + \boldsymbol{\eta}$, where b is the plaintext bit on the corresponding wire and $||\boldsymbol{\eta}||_\infty \le n \cdot m \cdot (N+1)^i$.

<u>Decrypt($\mathbf{s}, \mathbf{C}$).</u> Let $\mathbf{w} = -\lfloor q/2 \rfloor \cdot (0, \ldots, 0, 1) \in \mathbb{Z}_q^n$ be a scaled version of the n-th unit vector, and note that $\langle \mathbf{s}, \mathbf{w} \rangle = \lfloor q/2 \rfloor$ (since $s_n = -1$). The Decrypt procedure computes $z := [\mathbf{s} \times C \times G^{-1}(\mathbf{w})]_q$, outputting 0 if $|z| < q/4$ and 1 otherwise.

Correctness. To see that correctness holds, consider a ciphertext $\mathbf{C}$ at a level τ or below, so we have $\mathbf{sC} = b \cdot \mathbf{sG} + \boldsymbol{\eta}$ with $||\boldsymbol{\eta}||_\infty \le n \cdot m \cdot (N+1)^\tau$. Hence

$$z = \mathbf{s} \times C \times G^{-1}(\mathbf{w}) = (b \cdot \mathbf{sG} + \boldsymbol{\eta})G^{-1}(\mathbf{w}) = b \cdot \langle \mathbf{s}, \mathbf{w} \rangle + \left\langle \boldsymbol{\eta}, G^{-1}(\mathbf{w}) \right\rangle = b \cdot \lfloor q/2 \rfloor + v;$$

since $||\boldsymbol{\eta}||_\infty \le n \cdot m \cdot (N+1)^\tau$ and $G^{-1}(\mathbf{w})$ is a 0/1 vector then $|v| < n \cdot m \cdot (N+1)^{\tau+1} < q/4$, and therefore $|z| < q/4$ if $b = 0$ and $|z| > q/4$ if $b = 1$.

Lemma 5.3.6 (Semantic security). *The GSW scheme above is semantically secure under the DLWE*$[n, m, q, \alpha]$ *hardness assumption with* $\alpha = n/q$ *and* $m \ge 1 + 2n(2 + \log q)$.

Proof: Under the DLWE$[n, m, q, \alpha]$ hardness assumption, the public-key matrix $\mathbf{A}$ is pseudorandom in $\mathbb{Z}_q^{n\times m}$. It is sufficient therefore to prove that, if $\mathbf{A}$ was truly random, then the distribution of fresh encryption matrices would be statistically close to uniform in $\mathbb{Z}_q^{n\times N}$, regardless of the plaintext bit b.

Let $S = \{0,1\}^m$ and $L = \mathbb{Z}_q^n$; by Lemma 5.2.17 the distribution over $(\mathbf{A}, [\mathbf{Ar}]_q)$ for uniform $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ and $\mathbf{r} \leftarrow S$ is close to uniform over $\mathbb{Z}_q^{n\times(m+1)}$ up to statistical distance of

$$q^n \cdot \sqrt{2/|S|} \leq 2^{n\log q} \cdot 2^{(1-m)/2} \leq 2^{n\log q - n(2+\log q} = 2^{-2n}.$$

It follows that, except for probability 2^{-n} over the choice of $\mathbf{A}$, the distribution of $[\mathbf{Ar}]_q$ conditioned on $\mathbf{A}$ (defined over the random choice $\mathbf{r} \leftarrow \{0,1\}^m$) is 2^{-n}-close to uniform over $\mathbb{Z}_q^n$. Hence, the distribution of $[\mathbf{AR}]_q$ conditioned on $\mathbf{A}$ is $N2^{-n}$-close to uniform over $\mathbb{Z}_q^{n\times N}$, and therefore so is the distribution of the ciphertext matrix $\mathbf{C} = [b \cdot \mathbf{G} + \mathbf{AR}]_q$. ■

Parameters for the basic GSW scheme. The security and correctness arguments above rely on the conditions $m \geq 1 + 2n(2 + \log q)$ and $n \cdot m \cdot (N + 1)^{\tau+1} < q/4$, respectively, where $N = n \cdot \lceil \log q \rceil$. Substituting, we get the condition $q > 4n \cdot (1 + 2n(2 + \log q)) \cdot (1 + n\lceil \log q \rceil)^{\tau+1}$, which we can simplify as $q > (2n\log q)^{\tau+3}$.

On the other hand, we need to keep $q \leq 2^{n^\epsilon}$ for some $\epsilon < 1$ (else DLWE is no longer hard). This yields the condition $n^\epsilon > (\tau + 3)(\log n + \log\log q + 1)$, which we can again simplify (for large enough τ, n) as $n^\epsilon > 2\tau \log n$. One could check that these conditions are all satisfied by the following setting:

- $n = \max\{\lambda, \lceil (\frac{4}{\epsilon}\tau\log\tau)^{1/\epsilon} \rceil\}$
- $q = \lceil 2^{n^\epsilon} \rceil$,
- $m = 1 + \lceil 2n(2 + \log q) \rceil = O(n^{1+\epsilon})$, and
- $\alpha = n/q = n \cdot 2^{-n^\epsilon}$.

We note that, with this setting, the size of ciphertexts (fresh or evaluated) is polynomial in λ and τ, while the number of circuits in $\mathcal{C}_\tau$ is doubly exponential in τ, so the scheme is weakly compact as per Definition 5.2.9. Hence we have proved the following theorem:

Theorem 5.3.7. *Under the DLWE*$[n, m, q, \alpha]$ *hardness assumption with* $q = 2^{n^\epsilon}$, $\alpha = n \cdot 2^{-n^\epsilon}$, *and* $m = O(n^{1+\epsilon})$*, there exists a semantically-secure weakly-compact leveled homomorphic encryption scheme, where the complexity of evaluating each gate in a depth-d circuit is poly*$(\lambda, d^{1/\epsilon})$*.*

5.4 Realizing Fully Homomorphic Encryption

Above we described the basic GSW construction from [48], obtaining a leveled homomorphic encryption with security under the subexponential decision-LWE hardness assumption. In Section 5.4.1 below, we describe Gentry's *bootstrapping* technique [47] for transforming some leveled schemes into fully homomorphic ones, and then explain in Section 5.4.2 how to apply it to the GSW scheme. This yields either a leveled homomorphic encryption with security under the quasipolynomial (or even polynomial) decision-LWE hardness assumption, or alternatively a fully homomorphic scheme whose security depends on the same (quasi)polynomial DLWE in conjunction with a circular-security assumption.

5.4.1 Bootstrapping

So far we have a scheme $\mathcal{E}$ such that, for any bounded depth d, we can evaluate depth-d circuits by setting the parameters of $\mathcal{E}$ to size poly(d). To increase the homomorphic capacity of the scheme, however, we need to choose larger parameters, so also the complexity of encryption/evaluation/decryption increases. Gentry's insight in [47] was that this dependence can be broken if we manage to find a setting in which the homomorphic capacity is just slightly bigger than the decryption complexity.

We start by setting some notations. Fix some homomorphic encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ and a class $\mathcal{C} = \{\mathcal{C}_\tau\}$ of circuits with one output bit. Throughout this section we assume that all ciphertexts for parameters (λ, τ) have the same length (and hence the decryption procedure can be described by a single circuit).

- For $\lambda, \tau \in \mathbb{N}$, let $\mathcal{CT}_\mathcal{E}(\lambda, \tau)$ denote the set of all the fresh and evaluated ciphertexts that can result from evaluating circuits in $\mathcal{C}_\tau$:

$$\mathcal{CT}_\mathcal{E}(\lambda,\tau) \overset{def}{=} \left\{\mathsf{Encrypt}(\mathsf{pk}, b) : (\mathsf{pk},\mathsf{sk}) \in \mathsf{KeyGen}(1^\lambda, 1^\tau),\ b \in \{0,1\}\right\}$$
$$\cup \left\{\mathsf{Evaluate}(\mathsf{pk}, \Pi, \mathbf{c}) : \begin{array}{r} (\mathsf{pk},\mathsf{sk}) \in \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ \Pi \in \mathcal{C}_\tau,\ \mathbf{b} \in \{0,1\}^{\mathsf{inpLen}(\Pi)}, \\ \mathbf{c} \in \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}) \end{array}\right\}.$$

- Consider the decryption procedure for parameters λ, τ, and for any ciphertext $c \in \mathcal{CT}_\mathcal{E}(\lambda, \tau)$ denote by $D_c(\mathsf{sk}) \overset{def}{=} \mathsf{Decrypt}(\mathsf{sk}, c)$ the decryption circuit with c hardwired in. For any two ciphertexts $c_1, c_2 \in \mathcal{C}_\mathcal{E}(\lambda, \tau)$ let the *augmented decryption circuit* for these two ciphertexts be

$$D^*_{c_1,c_2}(\mathsf{sk}) \overset{def}{=} NAND(D_{c_1}(\mathsf{sk}),\ D_{c_2}(\mathsf{sk})).$$

Definition 5.4.1 (Bootstrappable encryption). *A homomorphic encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ is bootstrappable if its homomorphic capacity includes all the augmented decryption circuits. Specifically, there exists an (efficiently computable) polynomially bounded function $\tau(\cdot)$ such that, for every $\lambda \in \mathbb{N}$ and any $c_1, c_2 \in \mathcal{CT}_\mathcal{E}(\lambda, \tau(\lambda))$, we have $D^*_{c_1,c_2} \in \mathcal{C}_{\tau(\lambda)}$.*

Theorem 5.4.2 (Bootstrapping [47]). *Any bootstrappable homomorphic encryption scheme can be transformed into a compact leveled homomorphic encryption scheme.*

Proof: Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ be a bootstrappable homomorphic encryption scheme; we describe how to transform it into a compact leveled scheme $\mathcal{E}' = (\mathsf{KeyGen}', \mathsf{Encrypt}', \mathsf{Decrypt}', \mathsf{Evaluate}')$.

- $\mathsf{KeyGen}'(\mathbf{1}^\lambda, \mathbf{1}^\mathbf{d})$. Let $\tau = \tau(\lambda)$ (which is unrelated to the input parameter d). For $i = 0, 1, \ldots, d$ run the underlying key generation to get $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau)$, and for $i < d$ encrypt all the bits of the i-th secret key under the $i+1$-st public key, $\mathbf{c}_i^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}_{i+1}, \mathsf{sk}_i)$. The secret key of $\mathcal{E}'$ consists of all the sk_i's, and the public key consists of all the pk_i's and $\mathbf{c}_i^*$'s,
$$\mathsf{sk}' = (\mathsf{sk}_0, \ldots, \mathsf{sk}_d),\ \mathsf{pk}' = (\mathsf{pk}_0, \mathbf{c}_1^*, \mathsf{pk}_1, \mathbf{c}_2^*, \ldots, \mathbf{c}_{d-1}^*, \mathsf{pk}_d).$$
- $\mathsf{Encrypt}'(\mathsf{pk}', \mathbf{b})$. Encryption uses the first underlying public key, setting $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}_0, b)$ and outputting $(0, c)$ (with the tag 0 signifying that this is a fresh ciphertext).
- $\mathsf{Evaluate}'(\mathsf{pk}', \Pi, \mathbf{c})$. We assume w.l.o.g. that Π is made of NAND gates and is *leveled*; i.e., the two inputs to any gate at level i come from gates at level $i-1$. The $\mathsf{Evaluate}'$ procedure goes over the circuit in topological order from inputs to outputs; for every gate at level i with inputs $(i-1, c_1)$ and $(i-1, c_2)$, compute the description of the circuit $D_{c_1,c_2}(\cdot)$. Then use the underlying evaluation procedure to set $c' \leftarrow \mathsf{Evaluate}(\mathsf{pk}_i, D_{c_1,c_2}, \mathbf{c}_{i-1}^*)$ and use (i, c') as the output ciphertext of this gate. The $\mathsf{Evaluate}'$ procedure outputs the ciphertexts at the output gate of Π.
- $\mathsf{Decrypt}'(\mathsf{sk}', \mathbf{c}')$. On $c' = (i, c)$, use the i-th secret key with the underlying decryption procedure to output $b \leftarrow \mathsf{Decrypt}(\mathsf{sk}_i, c)$.

Correctness is shown by induction over Π; the fresh ciphertexts $\mathbf{c}_i^*$ are correct (relative to secret key sk_0) because the underlying scheme is correct for fresh ciphertexts, and similarly the ciphertexts encrypting the inputs to Π (relative to sk_0).

Consider now any gate at level i in Π, and assume by induction that the two ciphertexts at the input satisfy $c_1, c_2 \in \mathcal{CT}_\mathcal{E}(\lambda, \tau)$, and that both these ciphertexts are correct. Namely setting $pt_j \leftarrow \mathsf{Decrypt}(\mathsf{sk}_{i-1}, c_j)$ (for $j = 1, 2$), b_1, b_2 are indeed the input bits to this gate when Π is evaluated on $\mathbf{b}$.

Let $c' \leftarrow \mathsf{Evaluate}(\mathsf{pk}_i, D^*_{c_1,c_2}, \mathbf{c}_{i-1}^*)$. Since $c_1, c_2 \in \mathcal{CT}_\mathcal{E}(\lambda, \tau)$ then $D^*_{c_1,c_2} \in \mathcal{C}_\tau$ (since $\mathcal{E}$ is bootstrappable). As $\mathbf{c}_i^*$ is a fresh encryption of sk_{i-1} under pk_i, then by definition also $c' \in \mathcal{CT}_\mathcal{E}(\lambda, \tau)$, and moreover by correctness of $\mathcal{E}$ we have
$$\mathsf{Decrypt}(\mathsf{sk}_i, c') = D^*_{c_1,c_2}(\mathsf{sk}_{i-1}) = NAND(D_{c_1}(\mathsf{sk}_{i-1}), D_{c_2}(\mathsf{sk}_{i-1})) = NAND(b_1, b_2),$$
as needed. This completes the proof of correctness.

To see that $\mathcal{E}'$ is semantically secure we consider a sequence of hybrid experiments: For $k = 0, 1, \ldots, d$, let $\mathcal{H}_k$ be an experiment that proceeds just like the semantic security experiment of $\mathcal{E}'$ except for key generation: In $\mathcal{H}_k$, we use $\mathbf{c}_i^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}_{i+1}, \mathsf{sk}_i)$ as in the scheme for $i = 0, \ldots, k-1$, but for $i = k, \ldots, d-1$ we encrypt the all-zero string instead, setting $\mathbf{c}_i^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}_{i+1}, \mathbf{0})$.

We observe that $\mathcal{H}_d$ is the actual semantic security experiment, for which we need to prove that the advantage of the adversary is negligible in λ. For all k the only difference between $\mathcal{H}_k$ and $\mathcal{H}_{k-1}$ is in some ciphertexts that are encrypted under pk_k, and the corresponding sk_k is never used anywhere in the experiment (in particular it is not encrypted under pk_{k+1}), then by semantic security of the underlying $\mathcal{E}$ we know that the adversary's advantage in $\mathcal{H}_{k-1}$ is close to that of $\mathcal{H}_k$ up to a negligible difference. Taken together, this means that the advantage in $\mathcal{H}_0$ is close to that of $\mathcal{H}_d$ up to a negligible difference. But since sk_0 is not used anywhere in $\mathcal{H}_0$, then by semantic security of the underlying $\mathcal{E}$ the adversary in $\mathcal{H}_0$ only has advantage negligible in λ. We conclude that also in $\mathcal{H}_d$ the adversary only has negligible advantage, as needed. ■

Below we sometimes refers to homomorphic evaluation of decryption as the *recryption* procedure.

Remark 5.4.3. *In many homomorphic encryption schemes, the* Decrypt *procedure can be partitioned into a public* post-evaluation processing *phase that depends only on the public key, followed by the "actual decryption" that uses the secret key. We note that in this case the circuits D_c and $D^*_{c_1,c_2}$ can be thought of as having the postprocessed ciphertext hardwired and consisting of only the "actual decryption" phase. Alternatively we could think of the postprocessing phase as belonging to the* Evaluate *procedure.*

Also, the secret key can be preprocessed, independently of the ciphertext to be decrypted, in order to decrease the complexity of the "actual encryption"; this preprocessing can be thought of as part of KeyGen.

5.4.1.1 Fully Homomorphic Encryption

The above transformation yields a compact scheme, but not a fully homomorphic one, since we need a new secret/public key-pair for every level in Π. A natural variant of this transformation uses a single pair, and includes in the public key for $\mathcal{E}'$ also an encryption of the secret key of $\mathcal{E}$ under its own public key. It is clear that correctness still holds, but the security of the result can no longer be reduced to just semantic security of the underlying scheme $\mathcal{E}$. Rather, we now need to assume that $\mathcal{E}$ enjoys also *circular security*. Circular security can be defined in different ways; for our purposes we only need a weak version (taken from [23]) that requires semantic security to hold even given $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{sk})$.

Definition 5.4.4 (Weak circular security [23]). *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *be an encryption scheme (homomorphic or not), and let A be an adversary. The (weak)* circular-security advantage *of A w.r.t. $\mathcal{E}$ is defined as*

$$CircAdv_A^{\mathcal{E}}(\lambda) \overset{def}{=} \left| \Pr\left[A(\mathsf{pk}, \mathbf{c}^*, c) = 1 : \begin{array}{c} 1^\tau \leftarrow A(1^\lambda),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ \mathbf{c}^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathsf{sk}), c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 1) \end{array} \right] \right.$$
$$\left. - \Pr\left[A(\mathsf{pk}, \mathbf{c}^*, c) = 1 : \begin{array}{c} 1^\tau \leftarrow A(1^\lambda),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\tau), \\ \mathbf{c}^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathsf{sk}), c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0) \end{array} \right] \right| .$$

$\mathcal{E}$ is weakly circular secure if, for every PPT adversary A, the advantage $CircAdv_A^{\mathcal{E}}(\lambda)$ is negligible in λ.

Theorem 5.4.5. *Any weakly circular-secure bootstrappable homomorphic encryption scheme can be transformed into a compact fully homomorphic encryption scheme.*

The proof of Theorem 5.4.5 is very similar to Theorem 5.4.2, except that security follows directly from Definition 5.4.4.

5.4.2 The GSW Scheme Is Bootstrappable

Below we show that the GSW scheme from Section 5.3 above can be made bootstrappable. To this end, we need to analyze the complexity of the decryption procedure, and establish that it is within the homomorphic capacity of the scheme.

5.4.2.1 Decryption Complexity

Recall that the decryption formula for the GSW scheme is

$$\mathsf{Decrypt}(\mathbf{s}, \mathbf{C}) = \begin{cases} 0 & \text{if } |[\mathbf{s} \times \mathbf{C} \times \mathbf{w}^T]_q| < q/4 \\ 1 & \text{otherwise} \end{cases},$$

where $\mathbf{w}$ is some fixed vector that does not depend on the secret key. We think of the multiplication by $\mathbf{w}$ as a post-evaluation processing step (as mentioned in Remark 5.4.3), computing the vector $\mathbf{u} := [\mathbf{C} \times \mathbf{w}]_q$. The vector $\mathbf{u}$ is a Regev ciphertext (cf. Section 5.3.1.2) relative to the secret key $\mathbf{s}$, and the actual decryption consists of computing the integer inner product $z := \langle \mathbf{s}, \mathbf{u} \rangle$ and checking if $|[z]_q| < q/4$. It is well known that computing modular inner product is in $NC1$ (e.g., [63]). Hence, the actual decryption can be done using a circuit of depth logarithmic in the bitsize of $\mathbf{s}$ (which is $n \log q$), namely in depth $\Theta(\log n + \log\log q)$. Below we illustrate one such circuit. Denoting $k = \lceil z/q \rfloor$ (so $[z]_q = z - kq$), we observe that

$$[z]_{q/2} = \begin{cases} \quad [z]_q \quad = \quad z - 2k \cdot (q/2) & \text{if } |[z_q]| < q/4, \\ [z]_q \pm (q/2) = z - (2k \pm 1) \cdot (q/2) & \text{if } |[z_q]| > q/4. \end{cases}$$

Since $[z]_{q/2} = z - \lceil z/(q/2) \rfloor \cdot (q/2)$, it follows that comparing $|[z]_q|$ with $q/4$ can be done by checking if $\lceil z/(q/2) \rfloor$ is even or odd, namely by computing the bit $\lceil z/(q/2) \rfloor \bmod 2$.

To slightly simplify things, when setting the parameters for any depth-d homomorphism, we make the modulus q one bit larger than it strictly needs to be, thus ensuring that the eventual noise is bounded below $q/8$ (rather than $q/4$). This means that we have the guarantee that $|[z]_q|$ is either smaller than $q/8$ or larger than $3q/8$, so we can afford some error in the calculations without affecting the result of comparing with $q/4$. In particular, we have the guarantee that the value $z/(q/2)$ is within $1/4$ of an integer, so an error of less than $1/4$ in computing that value will not affect the way that it is rounded.

Let $\hat{q}$ be an approximation of the rational number $1/2q$ up to $t = 3 + \lceil \log n + 2 \log q \rceil$ bits of precision. Since $|\hat{q} - (1/2q)| < 2^{-t}$ and $|z| < nq^2$, then $|z\hat{q} - (z/2q)| < nq^2 \cdot 2^{-t} <$

$1/8$, and therefore $\lceil z/(q/2)\rfloor = \lceil z\hat{q}\rfloor$. Our decryption circuit is therefore constructed as follows:

- Still in the post-evaluation processing phase, after setting $\mathbf{u} = [\mathbf{C}\mathbf{w}]_q$, we compute the rational vector $\mathbf{r} = [\hat{q} \cdot \mathbf{u}]_2$, whose entries have t bits of precision.
- The actual decryption consists of computing the inner product $\langle \mathbf{s}, \mathbf{r}\rangle$ (over the rationals) and outputting the first bit to the left of the binary point (i.e., the most-significant bit of $[\langle \mathbf{s}, \mathbf{r}\rangle]_2$).

Each of the multiplications involved in the inner product $\langle \mathbf{s}, \mathbf{r}\rangle$ requires adding $\log q$ numbers, each of up to t bits. Summing up n such products, we therefore need to add $n \log q$ numbers, each of up to t bits.

To add these numbers, we use the 3-for-2 addition method (cf. [63]),[6] where the sum of three ℓ-bit numbers can be replaced by the sum of two numbers of up to $\ell + 1$ bits: Let x, y, z be three ℓ-bit numbers and consider the i-th bit position. The sum $x_i + y_i + z_i < 3$ can be represented by only two bits: the lower bit is the XOR of the three and the upper bit is their majority; i.e., we replace the three numbers x, y, z by the two numbers u, v, such that $u_i = XOR(x_i, y_i, z_i)$ and $v_{i+1} = MAJ(x_i, y_i, z_i)$.

Repeating this process recursively, after $\Theta(\log(n \log q))$ levels, we are left with only two t-bit numbers (since we can ignore all but the first bit to the left of the binary point). Adding these last two numbers can be done in logarithmic depth in the bit length t, using carry look-ahead. Here we are only interested in the carry bit into the first position to the right of the binary point, which can be expressed as $r := \bigvee_{i=0}^{t-1}\left(x_i \wedge y_i \bigwedge_{j=i+1}^{t-1}(x_j \vee y_j)\right)$. This expression has constant depth and fan-in $\Theta(t^2)$, so it can be computed by a fan-in-2 circuit of depth $\Theta(\log\ t)$. Hence we get total depth of $d = \Theta(\log t + \log(n \log q)) = \Theta(\log n + \log\log q)$.

Parameters for bootstrappable homomorphic encryption. To obtain a bootstrappable construction we need to set the parameters so that the complexity of (augmented) decryption stays within the homomorphic capacity of the scheme. By the analysis above, this means that we need to support evaluation of circuits of depth $\tau \le \rho(\log n + \log\log q)$ for some constant ρ.

Recall from Section 5.3.2 that to support depth-τ homomorphism we need $q \ge (2n \log q)^{\tau+3}$, and substituting the value τ from above we get the sufficient condition $q \ge (n \log q)^{\rho'(\log n + \log\log q)}$ for some other constant $\rho' < 2\rho$. For $n/\log n > 4\rho'$, we can meet all the conditions with the following setting:

- $n = \lambda$,
- $q = \left\lceil n^{4\rho' \log n}\right\rceil = 2^{\Theta(\log^2 n)}$,
- $m = 1 + \lceil 2n(2 + \log q)\rceil = \Theta(n \log^2 n)$, and
- $\alpha = n/q = 2^{-\Theta(\log^2 n)}$.

These parameters yield a bootstrappable scheme under the quasipolynomial DLWE assumption, and by Theorem 5.4.2 also a compact leveled scheme under the same

[6] As described in Remark 5.4.7 below, we can use modulus-switching and key-switching techniques to decrease the vector dimension and the size of q prior to decryption. Doing so we could get a bootstrappable scheme even when using less efficient addition methods.

assumption. If we also assume circular security, then using Theorem 5.4.5 we get a compact fully homomorphic scheme.

Theorem 5.4.6. *Under the DLWE$[n, m, q, \alpha]$ hardness assumption with $q = 2^{polylog(n)}$, $\alpha = 2^{-polylog(n)}$, and $m = n \cdot polylog(n)$, there exists a semantically secure compact leveled homomorphic encryption scheme. The public-key size needed for evaluating level-d circuits is linear in d, but the ciphertext size and the complexity of evaluating each gate is only poly(λ), independent of d.*

Further assuming circular security of the basic GSW scheme with these parameters, there exists a semantically secure compact fully homomorphic encryption scheme.

Remark 5.4.7. *The parameters above can be improved slightly using the modulus-switching and key-switching tricks from Section 5.3.1.4. To use modulus-switching we choose the secret key from the error distribution χ^n rather than uniformly at random from $\mathbb{Z}_q^n$, and use the result of Applebaum et al. [4] to argue that security is unaffected. We also include a key-switching gadget $\mathbf{A}'_{\mathbf{s}_n:\mathbf{s}^*}$ in the public key, where $\mathbf{s}^* \in \mathbb{Z}_q^{n'}$ with $n' \ll n$.*

Then during post-evaluation processing we switch the Regev ciphertext $\mathbf{u}$ relative to $\mathbf{s}_n$ to a lower-dimension ciphertext relative to $\mathbf{s}^*$, and then modulus-switch it to a smaller modulus $q' \ll q$. Then decryption needs to implement $[\langle \mathbf{s}^*, \mathbf{u}^* \rangle]_{q'} \overset{?}{<} q'/4$, which has lower complexity than the previous test $[\langle \mathbf{s}, \mathbf{u} \rangle]_q \overset{?}{<} q/4$.

Used judiciously, this technique lets us set $q' = poly(n)$ (whereas $q = quasipoly(n)$). This yields significant improvement in the recryption complexity, but by itself does not allow us to relax the necessary hardness assumption from quasipolynomial to polynomial DLWE. The reason is that, although decryption is evaluated relative to the smaller q', the public encryption key and fresh ciphertexts must still use the larger q. Relaxing the hardness assumption requires other techniques, as described next.

5.4.3 Homomorphic Encryption Under Polynomial DLWE

Is it possible to weaken the quasipolynomial DLWE assumption from above and get FHE under polynomial DLWE? At first glance, the answer seems to be negative: recryption seems to require circuit depth of at least $\log n$, and each level increases the noise by at least some small polynomial factor, so the accumulated noise (and hence the size of the modulus q and the resulting LWE approximation factor) is at least $\text{poly}(n)^{\log n}$.

Brakerski and Vaikuntanathan, however, observed in [20] that the asymmetric noise growth from Equations 5.1 and 5.4 provides a way out. Recall that, for two ciphertext matrices $\mathbf{C}_1, \mathbf{C}_2$ with associated plaintext bits b_1, b_2 and noise vectors $\boldsymbol{\eta}_1, \boldsymbol{\eta}_2$, the noise vector when multiplying $\mathbf{C}_1 \times G^{-1}(\mathbf{C}_2)$ has the form $\boldsymbol{\eta}' = b_1 \cdot \boldsymbol{\eta}2 + \boldsymbol{\eta}_1 \times G^{-1}(\mathbf{C}_2)$, so $\boldsymbol{\eta}_1$ has much greater influence on $\boldsymbol{\eta}'$ than $\boldsymbol{\eta}_2$. If we can ensure that we always keep $\boldsymbol{\eta}_1$ small, even if $\boldsymbol{\eta}_2$ is much bigger, then we can slow down the noise growth considerably.

As an illustrating example, consider multiplying a sequence of n ciphertexts, $\mathbf{C}_i$, $i = 1, \ldots n$, with associated plaintext bits b_i and noise vectors $\boldsymbol{\eta}_i$ of similar magnitude (say $\|\boldsymbol{\eta}_i\|_\infty \leq n$ for all i). This n-wise product can be implemented in a balanced binary tree, or using left- or right-associative trees (or any form in between), and these different strategies exhibit very different noise behavior. The balanced strategy is

$$\begin{aligned}\mathbf{C}_{\mathsf{bal}} = {} & \mathbf{C}_1 \times G^{-1}(\mathbf{C}_2) \times G^{-1}(\mathbf{C}_3 \times G^{-1}(\mathbf{C}_4)) \\ & \times G^{-1}(\mathbf{C}_5 \times G^{-1}(\mathbf{C}_6) \times G^{-1}(\mathbf{C}_7 \times G^{-1}(\mathbf{C}_8))) \cdots\end{aligned}$$

and its noise behavior is exactly what we analyzed above: We have a $\log(n)$-high binary tree of multiplications, and the noise magnitude at level i up the tree is roughly $m^i \cdot n$. Hence, the resulting matrix $\mathbf{C}_{bal}$ has a quasipolynomial noise magnitude of about $m^{\log n}$. The left-associative strategy is

$$\begin{tikzcd}
\mathbf{C}_1 \arrow[r] & \otimes \arrow[r] & \otimes \arrow[r] & \otimes \arrow[r] & \cdots \arrow[r] & \otimes \\
\mathbf{C}_2 \arrow[ur, "G^{-1}"'] & \mathbf{C}_3 \arrow[ur, "G^{-1}"'] & \mathbf{C}_4 \arrow[ur, "G^{-1}"'] & & \mathbf{C}_n \arrow[ur, "G^{-1}"'] &
\end{tikzcd}$$

$$\mathbf{C}_{\mathsf{left}} = (\cdots((\mathbf{C}_1 \times G^{-1}(\mathbf{C}_2)) \times G^{-1}(\mathbf{C}_3)) \cdots) \times G^{-1}(\mathbf{C}_n),$$

and its noise is much bigger. Specifically, let us denote by $\mathbf{C}_i^*, b_i^*, \boldsymbol{\eta}_i^*$ the ciphertext, plaintext, and noise after multiplying the leftmost i matrices. After the multiplication $\mathbf{C}_i^* \times G^{-1}(C_{i+1})$, the noise vector is $\boldsymbol{\eta}_{i+1}^* = b_i^* \boldsymbol{\eta}_{i+1} + \boldsymbol{\eta}_i^* G^{-1}(\mathbf{C}_i)$, which has magnitude of roughly $\|\boldsymbol{\eta}_{i+1}^*\|_\infty \approx \|\boldsymbol{\eta}_{i+1}\|_\infty + m \cdot \|\boldsymbol{\eta}_i^*\|_\infty \approx (1 + m + \cdots + m^i) \cdot n$. Hence, the overall noise magnitude is roughly $m^n \cdot n$, which is fully exponential in n.

On the other hand, the right-associative strategy is

$$\begin{tikzcd}
\mathbf{C}_{n-1} \arrow[dr] & & \mathbf{C}_3 \arrow[dr] & \mathbf{C}_2 \arrow[dr] & \mathbf{C}_1 \arrow[dr] & \\
\mathbf{C}_n \arrow[r, "G^{-1}"'] & \otimes \arrow[r] & \cdots \arrow[r, "G^{-1}"'] & \otimes \arrow[r, "G^{-1}"'] & \otimes \arrow[r, "G^{-1}"'] & \otimes
\end{tikzcd}$$

$$\mathbf{C}_{\mathsf{rght}} = \mathbf{C}_1 \times G^{-1}(\mathbf{C}_2 \times G^{-1}(\cdots \mathbf{C}_{n-1} \times G^{-1}(\mathbf{C}_n) \cdots)),$$

and this strategy has a much smaller noise than even the balanced one. Again let us denote by $\mathbf{C}_i^*, b_i^*, \boldsymbol{\eta}_i^*$ the ciphertext, plaintext, and noise after multiplying the matrices i through n. After incorporating also the $i-1$-st matrix, $\mathbf{C}_{i-1} \times G^{-1}(C_i^*)$, the noise vector is $\boldsymbol{\eta}_{i-1}^* = b_i \boldsymbol{\eta}_i^* + \boldsymbol{\eta}_{i-1} G^{-1}(\mathbf{C}_i^*)$, which has magnitude at most $\|\boldsymbol{\eta}_{i-1}^*\|_\infty \leq \|\boldsymbol{\eta}_i^*\|_\infty + m \cdot \|\boldsymbol{\eta}_{i-1}\|_\infty \leq n(m + m + \ldots + m)$. Hence, the overall noise magnitude is at most mn^2, which is only polynomial in n.

"Asymmetric circuits" and branching programs. To take advantage of the noise asymmetry, we need to design "asymmetric circuits" in which all the multiplications have the form $\mathbf{C} \times G^{-1}(\mathbf{C}^*)$, where $\mathbf{C}_1$ is a fresh encryption of an input bit (and hence has small noise) while $\mathbf{C}^*$ can be an evaluated ciphertext. One approach that naturally yields such circuits uses Barrington's theorem [7] and permutation branching programs.

Definition 5.4.8 (Permutation branching programs). *A width-w, length-ℓ permutation branching-program over inputs in $\{0,1\}^n$ consists of a sequence of ℓ tuples, $BP = ((\mathsf{inpLen}(i), \mathbf{A}_{i,0}, , \mathbf{A}_{i,1}) : i = 1 \ldots, \ell)$ where the $\mathbf{A}_{i,b}$'s are permutation matrices in $\{0,1\}^{w \times w}$, and $\mathsf{inpLen} : [\ell] \to [n]$ specifies the order in which input bits are examined (i.e., step i in the branching program examines the input bit $x_{\mathsf{inpLen}(i)}$). The function computed by this branching program is*

$$f_{BP}(x) \overset{def}{=} (\prod_{i=1}^{\ell} A_{i,x_{\mathsf{inpLen}(i)}})[1,1].$$

In words, $f_{BP}(x) = 1$ if composing all the permutations chosen by the bits of x in the different steps yields a permutation that maps 1 to itself, and otherwise $f_{BP}(x) = 0$.

Theorem 5.4.9 (Barrington's Theorem [7]). *If the function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a depth-d fan-in-2 binary circuit, then f can also be computed by a width-5, length-4^d permutation branching program.*

Barrington's theorem directly yields a polynomial-size asymmetric circuit for computing any $NC1$ circuit: We keep an "evaluated state" consisting of the current cumulative product $\prod_{i=j+1}^{\ell} A_{i,x_{\mathsf{inpLen}(i)}}$, then multiply this state on the left by the "fresh matrix" $\mathbf{A}'_j = x_k \mathbf{A}_{j,1} + (1 - x_k)\mathbf{A}_{j,0}$, with $k = \mathsf{inpLen}(j)$. Since the $\mathbf{A}_{i,b}$'s are all permutation matrices then all the intermediate values in this computation are always in $\{0,1\}$ (and moreover every sum computed during this computation always has exactly one term equaling 1 and all other terms equaling 0).

When evaluating this circuit on encrypted data, we get an encryption of $\mathbf{A}'_j$ that depends linearly on the fresh encryption of $x_{\mathsf{inpLen}(j)}$, and we multiply it (on the left) by the evaluated encryption of the cumulative product so far. When using the GSW scheme with parameters n, m, q, the cumulative noise at the end of the calculation is bounded by $4^d \cdot m \cdot n$, which is polynomial in n when the depth d is logarithmic. We can therefore evaluate the recryption function while only increasing the noise to magnitude polynomial in n, so we can set $q = \text{poly}(n)$ (and therefore $\alpha = 1/\text{poly}(n)$) and $m = O(n \log n)$.

Theorem 5.4.10. *Under the DLWE$[n, m, q, \alpha]$ hardness assumption with $q = poly(n)$, $\alpha = 1/poly(n)$, and $m = O(n \log n)$, there exists a semantically secure compact leveled homomorphic encryption scheme. The public-key size needed for evaluating level-d circuits is linear in d, but the complexity of evaluating each gate is only poly(λ), independent of d.*

Further assuming circular security of the basic GSW scheme with these parameters, there exists a semantically secure compact fully homomorphic encryption scheme.

Remark 5.4.11. *The key- and modulus-switching tricks from Remark 5.4.7 can be used here too, to reduce both the recryption complexity and the constant in the exponent of the polynomial-DLWE assumption. Brakerski and Vaikuntanathan describe in [20] how iterated application of these two tricks can be used to reduce the hardness assumption to DLWE with $\alpha = 1/n^{1/2+\epsilon}$ for any constant $\epsilon > 0$, which nearly* *matches the best setting $\alpha = 1/\tilde{O}(\sqrt{n})$ for any known lattice-based public-key encryption.*

The concrete efficiency of recryption was significantly improved by Alperin-Sheriff and Peikert [3], while still relying on the same DLWE with $\alpha = 1/n^{1/2+\epsilon}$. See Section 5.5.1 for more discussion of efficiency considerations.

5.4.4 Realizing Strong Homomorphism

The fully homomorphic scheme from above may still fail to provide strong homomorphism or even circuit privacy. Following Gentry [47], we can achieve statistical strong homomorphism using a Refresh procedure based on noise-flooding. A naive use of this technique would require increasing the parameters and using a stronger DLWE variant, but Ducas and Stehlé showed in [31] that iterated use of Refresh can avoid these limitations.

For the rest of this section, it will be convenient to assume that, the recryption procedure of the fully homomorphic encryption returns not the GSW ciphertext matrix $\mathbf{C}$ but rather the post-processed "Regev ciphertext" $\mathbf{u} = [\mathbf{Cw}]_q$, which is decrypted by computing $z := [\langle \mathbf{s}, \mathbf{u} \rangle]_q$ and comparing $|z|$ with $q/4$. (The distinction between these two forms of ciphertexts is only a matter of convenience, as we can always get back a GSW ciphertext by running the recryption procedure again without post-processing.) We denote running the recryption process on the post-processed $\mathbf{u}$ and returning the post-processed result by $\mathbf{u}' \leftarrow \mathsf{Recrypt}_{\mathsf{pk}}(\mathbf{u})$.

Rerandomizing a ciphertext. For a vector $\mathbf{s} \in \mathbb{Z}_q^n$, denote the linear subspace orthogonal to $\mathbf{s}$ by

$$L_{\mathbf{s}}^{\perp} \overset{def}{=} \{\mathbf{v} \in \mathbb{Z}_q : \langle \mathbf{s}, \mathbf{v} \rangle = 0 \pmod q\}.$$

Also denote the scaled unit vector by $\mathbf{w} \overset{def}{=} \lfloor q/2 \rfloor \cdot (0, \ldots, 0, -1)$, and the "radius-$\rho$ 1-dimensional ball" by

$$\mathcal{B}_\rho \overset{def}{=} \{x \cdot (0, \ldots, 0, 1) : \ x \in \mathbb{Z}, |x| \le \rho\}.$$

An easy-to-verify property that we use below is that, for $\rho', \rho \in \mathbb{N}$ and any $\mathbf{u} \in \mathcal{B}_{\rho'}$, the statistical distance between the uniform distributions on $\mathcal{B}_\rho$ and on $\mathbf{u} + \mathcal{B}_\rho$ is bounded by $\rho'/(2\rho + 1)$.

Since the last entry in a GSW secret key $\mathbf{s}$ is $s_n = -1$, then for $\rho < \lfloor q/4 \rfloor$ any vector $\mathbf{u} \in L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho$ is decrypted to zero and any vector $\mathbf{u} \in L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho + \mathbf{w}$ is decrypted to one.

Let $\mathbf{u}$ be some fixed vector in either $L_{\mathbf{s}}^{\perp} + \mathcal{B}_{\rho'}$ or $L_{\mathbf{s}}^{\perp} + \mathcal{B}_{\rho'} + \mathbf{w}$. We rerandomize $\mathbf{u}$ by adding to it a random element in $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho$ for some $\rho > \rho'$ such that $\rho + \rho' < \lfloor q/4 \rfloor$. The resulting vector $\mathbf{u}'$ still decrypts to the same bit as $\mathbf{u}$, and the statistical distance between the distribution of $\mathbf{u}'$ and the uniform distribution over $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho$ (or over $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho + \mathbf{w}$) is bounded by $\rho'/(2\rho + 1)$. If $\rho \gg \rho'$ then the distribution of $\mathbf{u}'$ is almost independent of the original vector $\mathbf{u}$, except for the plaintext bit that it decrypts to. To perform rerandomization, it is therefore sufficient to be able to choose a nearly uniform vector in $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho$. It turns out that the public-key matrix $\mathbf{A}$ is all we need for that purpose.

Lemma 5.4.12. *For any fixed vector* $\mathbf{s} \in \mathbb{Z}_q^n$ *with* $s_n = -1$, *denote by* $\mathcal{D}_{\mathbf{s}}$ *the distribution over public keys corresponding to the secret key* $\mathbf{s}$; *namely choosing the top* $n-1$ *rows at random* $\mathbf{A}' \leftarrow \mathbb{Z}_q^{(n-1)\times m}$ *and the last row as* $\mathbf{s}'\mathbf{A}' + \boldsymbol{\eta}$, *where* $\mathbf{s}'$ *are the first* $n-1$ *entries in* $\mathbf{s}$ *and* $\boldsymbol{\eta} \leftarrow \chi^m$.

Then with probability at least $1 - 2^{-n}$ *over the choice of* $\mathbf{A} \leftarrow \mathcal{D}_{\mathbf{s}}$ *and for all* $\rho \in \mathbb{N}$, *the distribution*

$$\mathcal{R}_{\mathbf{A},\rho} \stackrel{def}{=} \{\mathbf{Ar} + \mathbf{v} \ : \ \mathbf{r} \leftarrow \{0,1\}^m, \ \mathbf{v} \leftarrow \mathcal{B}_\rho\}$$

is close to the uniform distribution over $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho$, *up to statistical distance at most* $\frac{nm}{2\rho+1} + 2^{-n}$.

Proof: Let $\mathbf{s}'$ be the first $n-1$ entries of $\mathbf{s}$, $\mathbf{A}'$ be the first $n-1$ rows of $\mathbf{A}$, and $\tilde{\mathbf{A}}$ be the matrix $\mathbf{A}$ with the last row replaced by $\mathbf{s}'\mathbf{A}'$ (i.e., the last row of $\mathbf{A}$ with the error $\boldsymbol{\eta}$ removed). Then $\mathbf{s}\tilde{\mathbf{A}} = 0$, and moreover the columns of $\tilde{A} \in \mathbb{Z}_q^{n\times m}$ are uniform and independent in $L_{\mathbf{s}}^{\perp}$. By Lemma 5.2.17, with probability $1 - 2^{-n}$ over $\tilde{\mathbf{A}}$, the distribution of $[\tilde{\mathbf{A}}\mathbf{r}]_q$ conditioned on $\tilde{\mathbf{A}}$ is 2^{-n} away from uniform over $L_{\mathbf{s}}^{\perp}$.

Next observe that $\mathbf{Ar} = \tilde{\mathbf{A}}\mathbf{r} + \boldsymbol{\delta}$, where $\boldsymbol{\delta} = \langle \boldsymbol{\eta}, \mathbf{r}\rangle \cdot (0,\ldots,0,1)^t \in \mathcal{B}_{nm}$. We therefore have $\mathbf{Ar} + \mathbf{v} = \tilde{\mathbf{A}}\mathbf{r} + \delta + \mathbf{v}$, where $\tilde{\mathbf{A}}\mathbf{r}$ is 2^{-n}-close to uniform over $L_{\mathbf{s}}^{\perp}$, and regardless of the value of δ we have that $\delta + \mathbf{v}$ is $\frac{nm}{2\rho+1}$-close to uniform over $\mathcal{B}_\rho$. Hence $\mathbf{Ar} + \mathbf{v}$ is close to uniform in $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho$, up to $\frac{nm}{2\rho+1} + 2^{-n}$. ■

Corollary 5.4.13. *Let* $\rho, \rho' \in \mathbb{N}$, *and fix the two vectors* $\mathbf{s} \in \mathbb{Z}_q^n$ *with* $s_n = -1$ *and* $\mathbf{u} \in (L_{\mathbf{s}}^{\perp} + \mathcal{B}_{\rho'}) \cup (L_{\mathbf{s}}^{\perp} + \mathcal{B}_{\rho'} + \mathbf{w})$. *Then with probability at least* $1 - 2^{-n}$ *over the choice of* $\mathbf{A} \leftarrow \mathcal{D}_{\mathbf{s}}$, *the distribution* $\mathbf{u} + \mathcal{R}_{\mathbf{A},\rho}$ *is within statistical distance* $\frac{nm+\rho'}{2\rho+1} + 2^{-n}$ *of:*

- *The uniform distribution over* $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho$ *if* $\mathbf{u} \in L_{\mathbf{s}}^{\perp} + \mathcal{B}'_\rho$, *or*
- *The uniform distribution over* $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho + \mathbf{w}$ *if* $\mathbf{u} \in L_{\mathbf{s}}^{\perp} + \mathcal{B}'_\rho + \mathbf{w}$.

Strong homomorphic encryption. To obtain strong homomorphic encryption, we begin with the fully homomorphic scheme from above but modify the parameters, making q larger by some factor β (to be determined later) than what is needed to get full homomorphism. Specifically, we make q large enough to ensure that after recryption (and post-evaluation processing) the noise is bounded by some $\rho' < q/8 - \beta$; i.e., we always get a vector $\mathbf{u} \in (L_{\mathbf{s}}^{\perp} + \mathcal{B}_{\rho'}) \cup (L_{\mathbf{s}}^{\perp} + \mathcal{B}_{\rho'} + \mathbf{w})$. We then define a refresh procedure as

$$\mathsf{Refresh}_{\mathsf{pk}}(\mathbf{u}) = [\mathsf{Recrypt}_{\mathsf{pk}}(\mathbf{u}) + \mathcal{R}_{\mathbf{A},\beta}]_q.$$

We then modify both the encryption and evaluation procedures, making them output $\mathbf{u}' \leftarrow \mathsf{Refresh}_{\mathsf{pk}}(\mathbf{u})$ rather than the vector $\mathbf{u}$ as before. By Corollary 5.4.13 the output distribution of the new $\mathsf{Encrypt}$ and $\mathsf{Evaluate}$ is close to uniform over $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho$ or $L_{\mathbf{s}}^{\perp} + \mathcal{B}_\rho + \mathbf{w}$ (depending on the encrypted bit), where $\rho = \rho' + \beta < q/8$. Correctness is not affected since the parameters are set so that the noise after Refresh is still bounded below $q/8$.

The statistical distance from uniform is at most $\frac{nm+\rho'}{2\beta+1} + 2^{-n}$, so to get negligible distance we need $\beta > (nm + \rho') \cdot 2^{\omega(\log n)}$. This means that β (and therefore q) must be superpolynomial in n, so to get strong homomorphism in this way we need to assume hardness of superpolynomial DLWE.

Strong homomorphism from polynomial DLWE. It was observed by Ducas and Stehlé [31] that iterating Refresh can be used to go beyond superpolynomial DLWE. To show this, we use the following general lemma:

Lemma 5.4.14 (Iterated refresh [31]). *Let D be an arbitrary domain and let $f : D \to D$ be a randomized function. If for some $\delta < 1$ it holds that $SD(f(x_1), f(x_2)) < \delta$ for any $x_1, x_2 \in D$, then for every $k \in \mathbb{N}$ and $x_1, x_2 \in D$ we have $SD(f^k(x_1), f^k(x_2)) < \delta^k$.*

Proof: The proof is by induction on k. The basis $k = 0$ holds vacuously, so assume that it holds for k and we prove for $k + 1$. Let $x_1, x_2 \in D$, so by the induction hypothesis we have $SD(f^k(x_1), f^k(x_2)) < \delta^k$.

The distributions $f^k(x_1), f^k(x_2)$ can therefore be expressed as convex combinations $f^k(x_1) = (1 - \delta^k) \cdot \mathcal{D} + \delta^k \cdot \mathcal{D}'_1$ and $f^k(x_2) = (1 - \delta^k) \cdot \mathcal{D} + \delta^k \cdot \mathcal{D}'_2$ for the same $\mathcal{D}$ and different $\mathcal{D}'_i$'s. Namely for $i = 1, 2$, $y_i \leftarrow f^k(x_i)$ is obtained by choosing a bit $b \in \{0, 1\}$ with $\Pr[b = 1] = \delta^k$, then drawing $y_i \leftarrow \mathcal{D}$ if $b = 0$ and $y_i \leftarrow \mathcal{D}'_i$ if $b = 1$. It follows that

$$SD(f^{k+1}(x_1), f^{k+1}(x_2)) \le (1 - \delta^k) \cdot \underbrace{SD(f(\mathcal{D}), f(\mathcal{D}))}_{=0} + \delta^k \cdot \underbrace{SD(f(\mathcal{D}'_1), f(\mathcal{D}'_2))}_{\le\delta} \le \delta^{k+1},$$

as needed. ■

Applying Lemma 5.4.14 to the Refresh procedure from above, we can now set the parameters so that $\beta = 2nm \cdot \rho'$. With this setting, for every $\mathbf{u}, \mathbf{u}' \in \mathcal{CT}$ that encrypt the same bit, we get $SD(\mathsf{Refresh}_{\mathsf{pk}}(\mathbf{u}_1), \mathsf{Refresh}_{\mathsf{pk}}(\mathbf{u}_2)) < 1/2$, so after $\omega(\log n)$ iterations we get negligible statistical distance. Since ρ' can be set to poly(n) as per Section 5.4.3 above, then we get $q = \rho' + \beta = \rho' \cdot (1 + 2nm) = \text{poly}(n)$, so we can rely on the hardness of polynomial DLWE.

Theorem 5.4.15. *Under the DLWE$[n, m, q, \alpha]$ hardness assumption with $q = poly(n)$, $\alpha = 1/poly(n)$, and $m = O(n \log n)$, in conjunction with circular security of the basic GSW scheme with these parameters, there exists a semantically secure strong fully homomorphic encryption scheme.*

5.5 Advanced Topics

In this section we briefly discuss other aspects of homomorphic encryption. We begin in Section 5.5.1 by describing the active research for devising more practical homomorphic encryption, reducing the overhead of computing on encrypted data as compared with computing on plaintext data. In Section 5.5.2, we touch on some (mostly failed) attempts at realizing homomorphic encryption by means other than lattice-based cryptography. In Section 5.5.3, we discuss carrying out computation on encrypted data using models of computation other than circuits. Finally, in Section 5.5.4, we describe uses of techniques similar to the GSW homomorphic encryption scheme to obtain other functionalities, such as multikey homomorphism, homomorphic commitment and signatures, multilinear maps, and obfuscation.

5.5.1 Faster Homomorphic Encryption

The GSW cryptosystem from above is capable of evaluating arbitrary circuits, but at a steep price: Letting the *overhead* of a homomorphic scheme be the ratio of encrypted computation complexity to unencrypted computation complexity (using a circuit model of computation), it is not hard to see that as described above this scheme has a very large polynomial overhead: Each plaintext bit is encrypted by a matrix of dimensions at least $\tilde{\Omega}(\lambda) \times \tilde{\Omega}(\lambda)$ (and entries of size polylog(λ) bits). Each NAND operation in the basic GSW scheme requires the multiplication of two such matrices, which takes $\tilde{\omega}(\lambda^{2.3})$ even using the most asymptotically efficient algorithm. (Of course, to use bootstrapping we would need to implement full homomorphic description for every gate, which would drive the overhead much higher.)

A lot of work over the last few years was devoted to improving this overhead, both asymptotically and practically. Perhaps the most significant improvement comes from working over large extension rings rather than over the integers, relying on the ring-LWE hardness assumption (RLWE) [68].[7] Working over a large ring of extension degree d allows one to reduce the degree of the matrices involved to as little as $n = \lambda/d$ while maintaining hardness, so in particular when setting $d = \lambda$ we can get $n = 1$ and $m = O(\log q) = \tilde{O}(\lambda)$. This yields ciphertexts of bit-size $O(\log^2 q) = \tilde{O(\lambda)}$, and the complexity of implementing basic NAND (without bootstrapping) is similarly reduced to $\tilde{O(\lambda)}$. Hence we get a variant of the basic weakly compact scheme with overhead quasilinear in λ.

A second major improvement comes from packing multiple plaintext bits in every ciphertext. Specifically, Smart and Vercauteren observed in [86] that, since our "scalars" now live in some polynomial ring, the Chinese remainder theorem in that ring can be used to encode many bits in each scalar, yielding multiple "plaintext slots" where additions and multiplications are applied to each slot separately. Careful choice of parameters lets one get as many as $\ell = \tilde{\Omega}(\lambda)$ such plaintext slots, making the plaintext-to-ciphertext expansion ration polylogarithmic in λ, and allowing one to compute the same function on ℓ different inputs at the price of a single computation.

We note, however, that packing many bits in each plaintext scalar is in general incompatible with the Brakerski–Vaikuntanathan method of exploiting the asymmetric noise growth: Packed scalars typically have norm polynomial in d (the extension degree of the ring), even if only 0's and 1's are packed in the slots. Since the

[7] Specifically we use cyclotomic rings, since they have many desirable algebraic properties.

GSW noise depends also on the plaintext size, we have to content ourselves with multiplicative growth of noise for every multiplication operation. In fact, using the packing technique yields better results for the "second-generation schemes" such as those in [17, 14, 67, 13] than for the "third-generation" GSW scheme [48].

But we can go even beyond batching. It was observed in [68, 17, 86] that the automorphisms in the polynomial ring can be used to "rotate" the contents of the plaintext slots, and Gentry et al. show in [44] how to use these "rotations" to perform efficient routing of plaintext slots between successive levels of any arbitrary circuit. This technique is then used in [44] to perform an entire computation on "packed" ciphertexts, resulting in a scheme with only poly-logarithmic overhead even when computing a single function on a single input. This can be extended to bootstrapping, yielding a fully homomorphic scheme with polylogarithmic overhead. (Further optimizations for bootstrapping were described in [43, 2, 57].)

There has also been much work devoted to practical efficiency and implementing of homomorphic encryption. For example Gentry et al. reported in [45] on an implementation of the "second-generation" Brakerski–Gentry–Vaikuntanathan scheme (BGV) and its use for evaluating "real-world circuits" such as the AES encryption/decryption circuits, and that implementation was further optimized in the HElib library of Halevi and Shoup [56], who also implemented practical bootstrapping for packed ciphertexts in a matter of minutes [57]. Also, building on techniques from [20, 3], Ducas and Micciancio implemented bootstrapping for a (non-packed) GSW-like scheme in less than a second [30].

5.5.2 Other Attempts at Realizing Homomorphic Encryption

Attempts to realize homomorphic encryption go back to the dawn of public-key cryptography, when the concept was first proposed by Rivest, Adleman, and Dertouzos [81]. Below we sketch some directions that were explored over the years, even though as of yet none of these directions have panned out.

5.5.2.1 The Hidden-Ideal Paradigm

One natural approach is to construct a scheme that uses an algebraic ring R and an ideal $I \subset R$, and relies for security on the hardness of distinguishing random elements in I from random elements in R. The plaintext space of such a scheme is the quotient ring R/I, ciphertexts are (representations of) elements in R, and homomorphic addition and multiplication are just the '$+$' and '$\times$' operations in R.

This approach was implicit in many additive homomorphic schemes; it was made explicit by Fellows and Koblitz [32], who also suggested a concrete realization that they called Polly Cracker. That scheme uses the ring of multivariate polynomials over some field, $R = F[X_1, \ldots, X_n]$, and the ideal I is a set of polynomials that have common root, $p(s_1, \ldots, s_n) = 0$. Ciphertexts are polynomials $p \in R$, the root itself $\mathbf{s} = (s_1, \ldots, s_n)$ is the secret key, and decryption corresponds to evaluating $p(\mathbf{s})$.

There is a lot of freedom in choosing the concrete representation of elements and their probability distributions (note that these R, I are infinite). The challenge in designing such a scheme is to find a succinct representation that allows efficient sampling and computation of '+' and '$\times$', but where (at least) recovering $\mathbf{s}$ is hard. Many attempts to find such representations has been made, so far with no success; see [66] for a survey. We note that, even though no candidate construction so far have survived cryptanalysis, there is also no reason to think that no such candidate exists.

5.5.2.2 Homomorphic Encryption from Binary Codes

Following Gentry's blueprint for constructing homomorphic encryption schemes, some attempts were made to instantiate this blueprint using binary codes as opposed to integer lattices. The hope was, that since the problem of learning-parity-with-noise (LPN) is similar in some ways to the learning-with-errors (LWE) problem, then the techniques for constructing LWE-based homomorphic encryption would extend also to LPN. The main challenge is that LPN with its notion of Hamming distance provides a very narrow range for noise manipulation. Specifically, while in LWE-based constructions we can handle noise with Euclidean norm polynomial in the dimension, when it comes to Hamming distance the noise must be strictly smaller then the dimension.

A notable attempt to port Gentry's blueprint to the LPN-based setting was made by Bogdanov and Lee [9]. They described a construction that has noticeable decryption error probability, which is carefully controlled via evaluation of majority gates in conjunction with the linear decryption function. That construction was later broken by Gauthier et al. [36], and moreover Brakerski proved in [15] that the approach from [9] cannot work as-is. Specifically, he proved that a scheme which is capable of computing majority cannot have a learnable decryption function (such as a linear function), even if it has a significant decryption error probability. Although Brakerski's result does not rule out basing homomorphic encryption on LPN, it does say that the decryption procedure of that scheme must be at least "somewhat complicated", rather than the simple inner product of some LPN-based (non-homomorphic) schemes such as [1].

5.5.2.3 Homomorphic Encryption from Group Theory

An alternative approach, using concepts from group theory, was proposed by Nuida in [73]: For a group G (written multiplicatively), a subgroup $H \subset G$ is *normal* in G if $g^{-1}hg \in H$ for any $h \in H$ and $g \in G$. A simple scheme with plaintext space $\{0,1\}$ that uses such G, H represents 0 by a random $h \in H$ and 1 by a random $g \in G$. Homomorphic OR is implemented by the group operation $\mathsf{homOR}(g_1, g_2) := g_1 g_2$, and homomorphic AND is implemented by the commutator $\mathsf{homAND}(g_1, g_2) = g_1 g_2 g_1^{-1} g_2^{-1}$. This can be used to compute arbitrary functions by using deMorgan's laws to push negations to the inputs, and encrypting a bit b as a pair $(Enc(b), Enc(1-b))$.

Such a scheme would need to provide methods for choosing random elements in G and H, rely on the indistinguishability of H from G for security, and provide a trapdoor that enables distinguishing H from G as a secret key. Some candidate implementations of this approach are discussed in [73], and some earlier proposals were shown to be insecure. As for the hidden-ideal paradigm, here too we currently neither have a viable candidate nor know of a reason to think that such candidates are impossible.

5.5.3 Homomorphic Encryption for Other Models of Computation

Although circuits are a convenient and universal model of (classical) computation, other models have been considered as well, both weaker and stronger. Weaker models of computation were considered before Gentry's result, and they sometimes allow schemes based on different (non-lattice) hardness assumptions. Stronger models are also considered, as they could provide better efficiency for applications. Below we list a few such models, and what is known about them.

Truth-tables. Homomorphic encryption for truth-tables allows an encryptor to encrypt the index into a table, and an evaluator to compute from it an encryption of the content of the corresponding entry in the table. This is closely related to the notion of *single-server private information retrieval* (PIR) [65]. Indeed, a two-round single-server PIR protocol immediately yields a weakly compact *secret-key* encryption scheme which is homomorphic for truth-tables (cf. [62]).

Moreover, a PIR protocol can be transformed to a public-key scheme using an auxiliary public-key encryption scheme: The recipient chooses a public/secret key pair for the encryption scheme, the sender sends the PIR-client message and in addition also an encryption of the client's PIR-state s under the public key, and the evaluator forwards the encrypted state to the recipient together with the PIR reply. The recipient uses its secret key to decrypt and recover the SFE state s, and then uses the procedure $\mathsf{SFE3}$ with this state to recover $F(x,y)$. The result is a public-key encryption scheme capable of "evaluating" any table-lookup, and it is compact as long as the client's PIR state is short. We note that all PIR constructions have small client state, and this can be enforced generically by having the client use a pseudorandom generator (PRG) to derive its randomness, and using the PRG seed as the client state.

Since two-message PIR implies also public-key encryption [29, 49], we have that compact public-key homomorphic encryption for truth-tables can be realized from any two-message PIR protocol. For example, this yields realizations with security based on various factoring-related assumptions such as quadratic residuosity [52], N-th residuosity [77], and phi-hiding [22]. See [75] for a survey of single-server PIR.

Branching programs. Polynomial-size branching programs are a fairly strong model of computation, being able to evaluate at least the complexity class $NC1$. Ishai and Paskin described in [62] a weakly compact encryption scheme which is homomorphic for branching programs, using the Damgård–Jurik cryptosystem [28] whose security relies on the N-th residuosity assumption.

Turing machines and RAM. Although circuits are universal, and hence fully-homomorphic encryption for circuits can evaluate any function on encrypted data, other models of computation such as Turing machines or RAM computation can provide faster processing. It is therefore desirable to make the complexity of homomorphic evaluation as low as the Turing-machine complexity or the RAM-complexity of the evaluated function, as opposed to its circuit complexity. Unfortunately this is often not possible, for example it is clear that the table lookup function $f_T(i) = T[i]$ cannot be evaluated for an encrypted index in its RAM complexity of $O(1)$. Nonetheless, a significant body of recent work (such as [51] and [42]) has been devoted to finding cases where processing encrypted data with better than circuit complexity is possible.

Homomorphic quantum computations. Going beyond classical computations, one may wish to be able to apply quantum computations to an encrypted quantum state. Note that we are asking for more than simply a classical homomorphic-encryption scheme which is resistant to quantum attacks.[8] Instead, imagine trying to run Shor's algorithm [84] for factoring an encrypted integer. Being able to evaluate classical circuits on encrypted data is not enough here; we need to be able to evaluate also quantum gates.

A first step toward homomorphic quantum computation was recently taken by Broadbent and Jeffery [21], who described a quantum homomorphic encryption scheme for a restricted class of quantum circuits, assuming classical fully homomorphic encryption. Specifically, their scheme can handle circuits with unbounded number of Clifford-group gates but only a constant non-Clifford depth. (This is somewhat analogous to classical arithmetic circuits with unlimited additions but constant multiplication depth.)

5.5.4 Beyond Homomorphic Encryption

Powerful as it is, homomorphic encryption is just one of a number of new cryptographic primitives that were developed in the last decade using tools from lattice-based cryptography. Below we describe some other primitives that use similar tools.

5.5.4.1 Multikey Homomorphic Encryption

One limitation of homomorphic encryption is that it can only process encrypted data relative to one key. Many times, however, we want to be ale to process data that was encrypted relative to several different keys. For example, multiple parties, each with its own key, may upload their encrypted data to the cloud, and we want the cloud to aggregate this data and compute useful statistics on it. Of course, recovering the plaintext result would then depend on all the parties cooperating, each bringing its corresponding secret key. A homomorphic scheme that supports such processing is called *multikey homomorphic*.

[8] Since the learning-with-errors problem is assumed to be hard even for quantum computers, then so are all LWE-based homomorphic encryption schemes.

The concept of multikey homomorphic encryption, along with a concrete realization based on the NTRU cryptosystem [68], was first described by López-Alt et al. in [67]. One drawback of that scheme is that an upper bound on the number of parties must be known at key-generation time, since the parameters grow with the number of parties. (A similar realization is possible under LWE, but it only supports a constant number of parties.)

A different realization under LWE (or RLWE) was recently described by Clear and McGoldrick [25], and later significantly simplified by Mukherjee and Wichs [72]. These schemes can support an arbitrary number of parties, but they rely on a common reference string that must be known at key-generation time.

5.5.4.2 Homomorphic Commitments and Signatures

Although homomorphic encryption allows computing on encrypted data, it does not provide any integrity guarantees for the computed values. For example, in the client-server application in which the client encrypts its input x and the server evaluates on it a function f, the client has no guarantees that the alleged evaluated ciphertext was indeed produced by evaluating f on the encrypted x.

Verifying the integrity of remote computation is generally known as *verifiable computation*, and it is the subject of a very active research effort. See, e.g., [89] for a survey (focusing on the practical-oriented side of that work). Some useful tools in this area are homomorphic commitments and signatures, which can be constructed using techniques similar to those used in the GSW cryptosystem, as noted by Gorbunov et al. [55].

Homomorphic commitments are similar to homomorphic encryption, except that, in addition to the ciphertext-evaluation procedure $\mathsf{Evaluate}$, there is also a decommitment-evaluation procedure $\mathsf{deEvaluate}$ for computing on the corresponding randomness, which is used by the committer. The property of $\mathsf{deEvaluate}$ is that, whenever we have $\mathbf{c} = \mathsf{Encrypt}(\mathsf{pk}, \mathbf{b}; \mathbf{r})$ (where $\mathbf{r}$ is the randomness used for encryption) and $c' \leftarrow \mathsf{Evaluate}(\mathsf{pk}, \Pi, \mathbf{c})$, then computing $r' \leftarrow \mathsf{deEvaluate}(\mathsf{pk}, \Pi, \mathbf{c}, \mathbf{b}, \mathbf{r})$ we get $\mathsf{Encrypt}(\mathsf{pk}, \Pi(\mathbf{b}); r') = c'$. Thinking of the randomness $\mathbf{r}$ as being a decommitment string, this means that the committer can compute the randomness r' that would open the evaluated ciphertext c' to the plaintext $b' = \Pi(\mathbf{b})$. It is easy to see that the basic GSW scheme as described in Section 5.3.2.4 supports such a decommitment-evaluation routine, since whenever we have two ciphertexts $\mathbf{C}_i$ such that $\mathbf{s} \times \mathbf{C}_i = b_i \cdot (\mathbf{s} \times \mathbf{G}) + \boldsymbol{\eta}_i$, then their sum and product satisfy

$$\mathbf{s} \times (\mathbf{C}_1 \pm \mathbf{C}_2) = (b_1 \pm b_2) \cdot (\mathbf{s} \times \mathbf{G}) + (\boldsymbol{\eta}_1 \pm \boldsymbol{\eta}_2)$$
$$\text{and } \mathbf{s} \times (\mathbf{C}_1 \times G^{-1}(\mathbf{C}_2)) = (b_1 \cdot b_2) \cdot (\mathbf{s} \times \mathbf{G}) + (b_1 \cdot \boldsymbol{\eta}_2 + \boldsymbol{\eta}_1 \times G^{-1}(\mathbf{C}_2)),$$

and these noise terms can be computed efficiently by the committer.

In homomorphic signatures, a data originator uses its secret key to sign messages, and it publishes the vector of messages $\mathbf{b} = (b_1, \ldots, b_n)$ and the corresponding vector of signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$. A data processor, knowing $\mathbf{b}, \boldsymbol{\sigma}$, and the public key, can efficiently generate a short evaluated signature $\sigma_{\Pi,b'}$ on the pair

$(\Pi, \Pi(\mathbf{b}))$, and that signature can be verified using the public key (even without knowing the original data $\mathbf{b}$). Gorbunov et al. described in [55] a construction of homomorphic signatures from homomorphic commitments, in which verifying an evaluated signature $\sigma_{\Pi,b'}$ can be partitioned to an offline phase that depends only on Π and an online phase that depends also on b' and $\sigma_{\Pi,b'}$, such that the complexity of the online phase is independent of Π.

5.5.4.3 Functional Encryption, Obfuscation, and Multilinear Maps

Homomorphic encryption is in particular a secure encryption scheme. So while it is possible to compute on encrypted data (or with encrypted programs), the result is still encrypted and it takes the secret key to make sense of it. In many applications, however, we would like to process encrypted data or programs, and get (only) the result of the computation in the clear. For example, consider applying a spam filter to encrypted email: Although the content of the email messages should remain secret, we may want the mail server to learn the spam/no-spam bit *in the clear* so that it can forward to us only the non-spam messages.

For another example, imagine that we have a good model for predicting the risk of heart attack based on various indicators, and we want to release this model for use by the public. At the same time, we want to withhold the inner workings of this model, either due to intellectual property concerns, or because we need to protect the privacy of patient data that was used to devise this model. Here too, we may want the model itself to be encrypted, but anyone should be able to evaluate the model on their own indicators and get the result in the clear.

These examples illustrate typical uses of *functional encryption* (the first example) and *code obfuscation* (the second example), and a large body of research is devoted to studying these concepts. Functional encryption for simple functions can be constructed from pairing-based cryptography (e.g., [64, 83, 11, 74]), and some variants that support all functions but offer weaker security can be based on the hardness of LWE (e.g., [53, 54]). However, more is required to obtain fully secure functional encryption for all functions, or code obfuscation for any expressive class of functions. As of now, the only viable tool that we have for realizing these concepts are the so-called *cryptographic multilinear maps*.

Cryptographic multilinear maps were envisioned by Boneh and Silverberg [12], but were constructed for the first time only a decade later by Garg et al. [34]. On a high level, they enable evaluation of arithmetic circuits over a large field on "encrypted" data, getting in the clear the bit saying whether or not the result is equal to zero, without being able to "decrypt" any of the intermediate values. We currently have three candidate constructions for multilinear maps [34, 26, 39] (with some variations on each), all following the same high-level approach: Very roughly, they all begin with some homomorphic encryption scheme, and then publish a *defective secret key*, which allows testing for zero but not decryption.

To use such multilinear maps for obfuscation or functional encryption, one needs to randomize the computation so that no two intermediate values will ever be equal to each other, but where all the randomness can be canceled on the output wire so that the zero-test can be used to determine the output value. Such randomization techniques were found for $NC1$ circuits, and a bootstrapping technique using homomorphic encryption is used to extend these constructions to any polynomial-size circuits. Following Garg et al. [35], this approach was used in very many works; see, e.g., [60] for a survey.

5.6 Suggested Reading

Below are pointers to additional reading on related topics that are not covered in detail in this tutorial.

Multi-hop and circuit-private FHE. The connections with secure computation protocols with emphasis on multihop and function privacy (without compactness) were studied by Gentry et al. [47] in the semi-honest model. Their treatment was extended to the malicious adversary model by Ostrovsky et al. [76].

Second-generation FHE. A good survey that covers the basics of the second-generation FHE schemes was written by Vaikuntanathan [87], with more details given in the work of Brakerski et al. [17]. The techniques for reducing the plaintext-to-ciphertext overhead to polylogarithmic are described in the work of Gentry et al. [44], and many practical optimizations are described in the work of Halevi and Shoup [56]. The *scale-invariant* flavor of second-generation schemes was introduced by Brakerski [14] and used also in the work of Bos et al. [13].

Third-generation FHE. The GSW cryptosystem was presented by Gentry et al. in [48], together with some extensions such as identity-based FHE. The use of asymmetric circuits was proposed by Brakerski and Vaikuntanathan [20], and additional bootstrapping optimizations using this approach were described by Alperin-Sheriff and Peikert [3] and by Ducas and Micciancio [30].

Multikey FHE. The concept of a multikey FHE was introduced by López-Alt et al. in [67], along with a solution based on NTRU. A construction based on LWE was first described by Clear and McGoldrick [25], and later significantly simplified by Mukherjee and Wichs [72] and improved further by Peikert and Shiehian [79].

Homomorphic commitments and signatures. An interesting usage of techniques very similar to those described in this tutorial for the purpose of homomorphic commitments and signatures was described by Gorbunov et al. in [55].

Acknowledgements I drew on many sources for this tutorial, most extensively on Craig Gentry's PhD thesis [37], a survey by Vinod Vaikuntanathan [87], and a blog by Boaz Barak and Zvika Brakerski [6]. I would like to thank Craig Gentry for teaching me most of what I know about FHE, and also the many other people with whom I collaborated on work in this area. Special thanks is due to Yehuda Lindell for the initiative to write this book.

References

[1] M. Alekhnovich. More on average case vs approximation complexity. *Computational Complexity*, 20(4):755–786, 2011. Extended abstract in FOCS 2003.

[2] J. Alperin-Sheriff and C. Peikert. Practical bootstrapping in quasilinear time. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013.

[3] J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314. Springer, 2014.

[4] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.

[5] S. Arora and R. Ge. New algorithms for learning in presence of errors. In L. Aceto, M. Henzinger, and J. Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2011.

[6] B. Barak and Z. Brakerski. The Swiss Army knife of cryptography. Blog document, accessed January 2016, http://windowsontheory.org/2012/05/01/the-swiss-army-knife-of-cryptography/, 2012.

[7] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[8] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993.

[9] A. Bogdanov and C. H. Lee. Homomorphic encryption from codes. IACR Cryptology ePrint Archive, Report 2011/622, 2011. http://eprint.iacr.org/2011/622.

[10] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In J. Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[11] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In Y. Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[12] D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. Cryptology ePrint Archive, Report 2002/080, 2002. http://eprint.iacr.org/2002/080.

[13] J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In M. Stam, editor, *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2013.

[14] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[15] Z. Brakerski. When homomorphism becomes a liability. In *TCC*, pages 143–161, 2013.

[16] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science (ITCS'12)*, 2012. Available at http://eprint.iacr.org/2011/277.

[17] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):13, 2014.

[18] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 575–584, New York, NY, USA, 2013. ACM.

[19] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[20] Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In M. Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 1–12. ACM, 2014.

[21] A. Broadbent and S. Jeffery. Quantum homomorphic encryption for circuits of low T-gate complexity. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 609–629. Springer, 2015.

[22] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6,*

1999, Proceeding, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.

[23] D. Cash, M. Green, and S. Hohenberger. New definitions and separations for circular security. In M. Fischlin, J. Buchmann, and M. Manulis, editors, *Public Key Cryptography - PKC 2012*, volume 7293 of *Lecture Notes in Computer Science*, pages 540–557. Springer Berlin Heidelberg, 2012.

[24] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[25] M. Clear and C. McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 630–656. Springer, 2015.

[26] J. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2013.

[27] J. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 487–504. Springer, 2011.

[28] I. Damgård, M. Jurik, and J. B. Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*, 9(6):371–385, 2010.

[29] G. DiCrescenzo, T. Malkin, and R. Ostrovsky. Single database private information retrieval implies oblivious transfer. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 2000.

[30] L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. 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.

[31] L. Ducas and D. Stehlé. Sanitization of FHE ciphertexts. Manuscript, Available from `https://heat-project.eu/School/Damien%20Stehle/HEAT_FHE_Stehle.pdf`, 2016.

[32] M. Fellows and N. Koblitz. Combinatorial cryptosystems galore! *Contemporary Mathematics*, 168:51–51, 1994.

[33] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[34] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[35] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[36] V. Gauthier, A. Otmani, and J.-P. Tillich. A distinguisher-based attack of a homomorphic encryption scheme relying on Reed-Solomon codes. Cryptology ePrint Archive, Report 2012/168, 2012. http://eprint.iacr.org/2012/168.

[37] C. Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009. AAI3382729.

[38] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

[39] C. Gentry, S. Gorbunov, and S. Halevi. Graph-induced multilinear maps from lattices. In Y. Dodis and J. B. Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527. Springer, 2015.

[40] C. Gentry and S. Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In R. Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 107–109. IEEE Computer Society, 2011.

[41] C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology - EUROCRYPT'11*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.

[42] C. Gentry, S. Halevi, M. Raykova, and D. Wichs. Outsourcing private RAM computation. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 404–413. IEEE Computer Society, 2014.

[43] C. Gentry, S. Halevi, and N. P. Smart. Better bootstrapping in fully homomorphic encryption. In M. Fischlin, J. A. Buchmann, and M. Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany,*

May 21-23, 2012. Proceedings, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012.

[44] C. Gentry, S. Halevi, and N. P. Smart. Fully homomorphic encryption with polylog overhead. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012. Available from http://eprint.iacr.org/2011/566.

[45] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

[46] C. Gentry, S. Halevi, and V. Vaikuntanathan. *i*-hop homomorphic encryption and rerandomizable Yao circuits. In T. Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2010. http://eprint.iacr.org/2010/145.

[47] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2010.

[48] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013, Part I*, pages 75–92. Springer, 2013.

[49] Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan. The relationship between public key encryption and oblivious transfer. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 325–335. IEEE Computer Society, 2000.

[50] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In A. V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[51] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run Turing machines on encrypted data. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553. Springer, 2013.

[52] S. Goldwasser and S. Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[53] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2012.

[54] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, 2015.

[55] S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In R. A. Servedio and R. Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 469–477. ACM, 2015.

[56] S. Halevi and V. Shoup. Algorithms in HElib. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2014.

[57] S. Halevi and V. Shoup. Bootstrapping for HElib. In E. Oswald and M. 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 641–670. Springer, 2015.

[58] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, Mar. 1999.

[59] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In J. Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[60] M. Horváth. Survey on cryptographic obfuscation. Cryptology ePrint Archive, Report 2015/412, 2015. http://eprint.iacr.org/2015/412.

[61] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In C. Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 433–442. ACM, 2008.

[62] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography - TCC'07*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[63] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. A)*, pages 869–941. MIT Press, Cambridge, MA, USA, 1990.

[64] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *J. Cryptology*, 26(2):191–224, 2013.

[65] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 364–373. IEEE Computer Society, 1997.

[66] F. Levy-dit-Vehel, M. G. Marinari, L. Perret, and C. Traverso. A survey on Polly Cracker systems. In *Gröbner Bases, Coding, and Cryptography*, pages 285–305. Springer, 2009.

[67] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In H. J. Karloff and T. Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1219–1234. ACM, 2012.

[68] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013.

[69] C. A. Melchor, P. Gaborit, and J. Herranz. Additively homomorphic encryption with d-operand multiplications. In T. Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2010.

[70] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, 2012.

[71] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

[72] P. Mukherjee and D. Wichs. Two round mutliparty computation via multi-key FHE. Cryptology ePrint Archive, Report 2015/345, 2015. http://eprint.iacr.org/2015/345, accessed Jan, 2016.

[73] K. Nuida. Candidate constructions of fully homomorphic encryption on finite simple groups without ciphertext noise. Cryptology ePrint Archive, Report 2014/097, 2014. http://eprint.iacr.org/2014/097, accessed Jan 2016.

[74] A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/2010/556.

[75] R. Ostrovsky and W. E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In T. Okamoto and X. Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference*

on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings, volume 4450 of *Lecture Notes in Computer Science*, pages 393–411. Springer, 2007. Available at http://eprint.iacr.org/2007/059.

[76] R. Ostrovsky, A. Paskin-Cherniavsky, and B. Paskin-Cherniavsky. Maliciously circuit-private FHE. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 536–553. Springer, 2014. Available from https://eprint.iacr.org/2013/307.

[77] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[78] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In M. Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342. ACM, 2009.

[79] C. Peikert and S. Shiehian. Multi-key FHE from LWE, revisited. Cryptology ePrint Archive, Report 2016/196, 2016. http://eprint.iacr.org/.

[80] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[81] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[82] R. Rothblum. Homomorphic encryption: From private-key to public-key. In Y. Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2011.

[83] A. Sahai and B. Waters. Slides on functional encryption. PowerPoint presentation, 2008. http://www.cs.utexas.edu/~bwaters/presentations/files/functional.ppt.

[84] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[85] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

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

[87] V. Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In R. Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 5–16. IEEE Computer Society, 2011.

[88] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 24–43, 2010.

[89] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015.

[90] A. C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science – FOCS '82*, pages 160–164. IEEE, 1982.